UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO
FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA
ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS

TESIS

USO DE TÉCNICAS DE MACHINE LEARNING PARA LA CLASIFICACIÓN DE IMÁGENES DE DANZAS TÍPICAS DEL CUSCO

Para optar al título profesional de:
INGENIERO INFORMÁTICO Y DE SISTEMAS

Presentado por:
Br. EDGAR RODOLFO QUISPE CONDORI
Asesor :
M.Sc. LAURO ENCISO RODAS
Co-asesor :
M.Sc. BERTHIN S. TORRES CALLAÑAUPA

CUSCO - PERÚ
2017

Este trabajo se lo dedico a mi familia,
en especial a mi madre y tío Jose.
Por apoyarme y la motivarme a ser mejor.

Resumen

Cusco es una de las ciudades del mundo con mayor presencia de danzas autóctonas. Cada año, antropólogos folkloristas encuentran más información sobre las danzas y así rescatan parte de nuestra cultura. Sólo en las festividades podemos ubicar al menos 4 danzas típicas muy comunes: Qhapaq Qolla, Qhapaq Chuncho, Negrillo, y Contradanza. Por otro lado, Cusco también es conocido como una de las ciudades más turísticas del mundo y con todo el avance tecnológico que se viene dando, ahora es más fácil tomar fotografías y así preservar diversos eventos. Por lo tanto, se cuenta con mucha información visual que no es procesada.

Computacionalmente hablando, se puede lidiar con este problema mediante la clasificación automática de imágenes. La aplicación de este problema puede ser visto en diferentes áreas como *Sistemas de vigilancia, sistemas de recuperación, compresión, segmentación de imágenes*, y muchos otras más. La clasificación de imágenes también representa un gran desafío debido a los problemas en iluminación, rotación, escala, oclusión o variación de puntos de vista. Actualmente, los trabajos en clasificación de danzas se enfocan en videos, sin embargo, en imágenes este problema está muy poco estudiado. Los trabajos más relacionados que usan imágenes buscan clasificar eventos culturales; a diferencia de éstos, la información contextual de la imagen no juega un papel relevante como para los eventos culturales.

Este trabajo busca clasificar imágenes de danzas típicas del Cusco, el problema se torna particularmente interesante debido a que además de los problemas ya mostrados, se debe trabajar sobre la figura humana. El método planteado fue dividido en dos etapas, en la primera se localiza al danzante dentro de la imagen mediante la detección de máscaras usando Histogram of Oriented Gradient y técnicas de detección de rostros, y en la segunda se realiza la clasificación mediante Bag-of-Words. El principal aporte de este trabajo es realizar un estudio del desempeño de técnicas de *Machine Learning* en la clasificación de imágenes de danzas, igualmente se creó un *dataset* con imágenes etiquetadas de las danzas de la región que fueron de interés para el trabajo. Cabe recalcar que este *dataset* es el primero de su tipo y que será puesto de manera abierta a la comunidad científica para aportar al estado-del-arte. Los resultados obtenidos muestran una tasa de acierto alta para éste problema.

Palabras Claves: *Clasificación de Danzas Típicas, Detección de Danzante, Viola & Jones, Histogram of Oriented Gradient, Adaboost, Random Forest, Support Vector Machine, Bag-of-Words.*

Abstract

Cusco is one of the cities of the world with greater presence of autochthonous dances. Every year, folklorist anthropologists find more information about dances and thus rescue part of our culture. Just at the festivities we can find at least 4 typical dances: Qhapaq Qolla, Qhapaq Chuncho, Negrillo, and Contradanza. On the other hand, Cusco is also known as one of the most touristic cities in the world and with all the technological advances that have been taking place, it is now easier to take photographs and preserve different events. Therefore, there is a lot of visual information that is not processed.

Computationally speaking, you can deal with this problem through the automatic classification of images. The application of this problem can be seen in different areas such as *surveillance systems, image retrieval systems, image compression, image segmentation*, and many others. The classification of images also represents a great challenge due to the problems in illumination, rotation, scale, occlusion or variation of points of view. Currently, the works in dance classification use video but not images. The most similar works that use images seek to recognize Cultural Events. Unlike this, the image context is not relevant for the results.

This work aims to classify Cusco typical dance images, this problem turns particularly interesting because we have to work over human shape. The proposed method is divided in two steps, the first detects dancers location based its masks and the second classifies using the Bag-Of-Words Approach. The main contribution of this work is a study of the performance of Machine Learning on dance classification, a dataset of the dances used in this work has been also built, this will be open access to contribute to the state-of-art. The results show a high accuracy in this problem.

Keywords: *Typical Dance Classification, Dancer Detection, Viola & Jones, Histogram of Oriented Gradient, Adaboost, Random Forest, Support Vector Machine, Bag-of-Words.*

Índice

Índice de Figuras

Índice de Tablas

Términos y Abreviaciones

Adaboost: Es un algoritmo de clasificación que combina clasificadores más débiles.

Árbol (Tree): Es un grafo sin ciclos donde para cada par de nodos existe un único camino.

Bag of Words(BoW): Es un modelo de clasificación que se basa en los histogramas generados a partir de los vectores de características de las imágenes.

Binning: Proceso de asignar un elemento a una de las barras de un histograma.

Bordes: Puntos de la imagen en los cuales existe un cambio de iluminación o existe discontinuidad.

Coding: Proceso de convertir un conjunto de datos a una representación más compacta, por ejemplo un histograma.

Cross-Validation: Método de evaluación para la clasificación, éste divide los datos en un conjunto de entrenamiento y evaluación.

Dataset: Conjunto de datos con características o propiedades similares.

Decision Tree: Es un predictor que se basa en la estructura de un árbol, cada nodo representa un test o pregunta y las hojas indican la salida.

Eigenfaces: Método de reconocimiento de rostros planteado por [Pentland et al., 1994].

Entropía: Cantidad de información promedio existente en un dato.

Exactitud (Recall): En estadística es considerado el sesgo de una estimación.

Filtros: Es una técnica en la cual se modifican el tamaño, color y sombra de una imagen.

Fisherfaces: Método de reconocimiento de datos presentado por [Fisher, 1936].

Gradiente: Un cambio del valor de la cantidad dentro de una variable.

Grid Search: Proceso de búsqueda de parámetros para un Support Vector Machine (SVM).

Histograma: Representación de variables donde la superficie de cada barra es proporcional a la frecuencia de los valores representados.

Histogram of Oriented Gradient (HOG): Es un método usado en visión computacional y procesamiento de imágenes para el propósito de detección de objetos.

HOG Pyramid: Extensión basada en HOG para la detección de humanos. Se basa en la detección a diferentes escalas.

Hough Transform: Algoritmo usado para la detección de formas geométricas dentro de las imágenes.

Integral Image: Representación de una imagen binaria que permite calcular de operaciones en tiempo constante O(1).

Kernel: Función usada para evaluar la similaridad de vectores de características.

k-Means: Algoritmo de aprendizaje no supervisado usado para la clusterización de datos.

k-Nearest Neighbors (kNN): Algoritmo de Machine Learning usado en la clasificación y regresión.

Local Binary Pattern (LBP): Descriptor de textura planteado por [Ojala et al., 2002] usado en visión computacional para la clasificación.

Pooling: Proceso de combinación de datos, es también conocido como subsampling. En el caso de Bag of Words es el proceso de combinación de histogramas.

Precisión (Precision): Dispersión del conjunto de valores obtenidos de mediciones repetidas de una magnitud.

Principal Component Analysis (PCA): Es un proceso estadístico que usa la transformación ortogonal para convertir un conjunto de observaciones en un conjunto de valores linealmente no correlacionados.

Puntos de Interés: Áreas de un imagen que contienen información relevante de la misma.

Random Forest: Algoritmo de Machine Learning usado en la clasificación y regresión.

Samples: Término utilizado para los datos de entrenamiento de Viola & Jones Cascade.

Scale-invariant feature transform (SIFT): Es un método que calcula puntos de interés de una imagen.

Sliding Windows: Proceso de recorrer una imagen mediante una región comúnmente de tamaño fijo.

Speeded up robust features (SURF): Es un método que calcula puntos de interés de una imagen.

Support Vector Machine (SVM): Algoritmo de Machine Learning usado en la clasificación.

Textura: Es un conjunto de métricas calculadas sobre una imagen que busca caracterizar la disposición espacial de color e intensidad.

Thresholding: Proceso de binarizar una imagen en base a un umbral.

Viola & Jones Cascade: Algoritmo planteado por [Viola and Jones, 2001] para la detección de rostros.

Capítulo 1

Generalidades

1.1 Aspectos Generales

Las generalidades del proyecto de investigación son presentadas en el presente capítulo. A continuación, se describe el problema así como el contexto en el cual se desarrolla el proyecto.

1.1.1 Problema de Investigación

1.1.1.1 Descripción del Problema

La danza forma parte de las actividades cotidianas, es una expresión de lo que es y cree una sociedad. El Perú es un país con una gran diversidad cultural, sólo en la fiesta de la Virgen del Carmen de Paucartambo se presentan 19 danzas. Esta gran variedad hace que una persona desconozca las danzas que son bailadas en su propia región, a eso se acopla la gran cantidad de personas que hacen turismo y que desconocen totalmente de las costumbres y expresiones artísticas del Perú.

Segun datos oficiales del Ministerio de Comercio Exterior y Turismo, el Perú recibió la visita de 3'110,020 de turistas internacionales durante los 10 primeros meses del 2016, ésto genera necesidades de todo tipo, incluyendo las tecnológicas. El desarrollo de trabajos que permitan tratar de manera automática toda la información que es generada y que a la vez permitan a todos los visitantes y locales aprender sobre la cultura de la región genera una necesidad que debe ser suplida.

El problema de clasificación de imágenes busca etiquetarlas de acuerdo a su contenido, comúnmente dentro de un conjunto de categorías predefinidas. Por ejemplo: Dada una imagen ¿Contiene ésta un perro o no?. Una de las principales aplicaciones de la clasificación de imágenes es *image retrieval*, en ella se realizan búsquedas dentro de un conjunto de datos para recuperar aquellas imágenes con un contenido visual en particular.

En el caso de imágenes de danzas el problema de clasificación se torna interesante debido a las diferentes variantes que puede existir sobre una misma danza. Por ejemplo, las posturas en las cuales se encuentra el danzante varían, éste se puede encontrar parado, agachado, sentado, saltando, etc (Figura 1). Ésto hace que un enfoque que clasifique las imágenes a partir de las posturas que toman los bailarines como parte de la danza no garantice un resultado eficaz.

Figura 1: *Ejemplos de posturas que puede presentar un danzante.*

Si bien la diferencia de atuendos entre danzas distintas es notable, existe también diferencias dentro de una misma danza, estos cambios son en su mayoría en los colores de la ropa que es usada. Ésto debido a que una misma danza es bailada en diferentes lugares o ya sea por la misma naturaleza de la danza (figura 2).

Figura 2: *Diferencias entre los colores de camisa dentro de la danza Qhapaq Qolla.*

En algunos casos, como la danza Qhapaq Chuncho, la variación de atuendos es más clara debido a que las costumbres son diferentes en cada provincia del Cusco (figura 3).

Figura 3: *De izquierda a derecha. Qhapaq Chuncho de Paucartambo, Qhapaq Chuncho de Huarocondo*

Otro problema son las variaciones de los puntos desde los cuales se pueden tomar una imagen: desde arriba o desde el suelo, a cuerpo completo o parcial, de algún adorno de la vestimenta en específico, de las máscaras, de la parte posterior del danzante o con oclusiones debido a algún objeto que cubre parcialmente al danzante, se puede presentar incluso que sobre una misma imagen haya más de un tipo de danza.

Además existen otros problemas generales de la clasificación de imágenes, entre ellos la iluminación, el ruido y resolución (figura 4) [1].

Figura 4: *De izquierda a derecha. Problemas de iluminación, ruido y resolución.*

Como se puede ver, el problema de clasificación de imágenes de danzas es bastante complejo debido a todas las variaciones que pueden presentar una imagen y la naturaleza de las danzas.

[1] La imagen de Qhapaq Qolla (problemas de calidad de imagen) ha sido extraída de `http://goo.gl/SB6XKK`

1.1.1.2 Formulación del Problema

Actualmente no se puede clasificar de manera automática danzas típicas de la región a partir de una imagen.

1.1.2 Antecedentes

De la bibliografía revisada se ve que actualmente no existe ningún trabajo que busque clasificar imágenes de danzas típicas, pero sí existen trabajos relacionados que buscan clasificar videos de danzas típicas, detectar eventos sociales y reconocer imágenes de eventos culturales.

1.1.2.1 Clasificación de Videos de Danzas Típicas

Si bien la clasificación danzas típicas en imágenes es relativamente nueva, en **videos** es un problema bastante estudiado. El problema de clasificación de danzas es planteado de manera puntual es el trabajo de [Samanta et al., 2012], este trabajo usa una técnica basada en representación esparza para danzas de la India. El primer paso es representar cada *frame* del video de la danza mediante un descriptor de posición basado en *histogram of oriented optical flow (HOOF)*, luego se aprende las poses base mediante un diccionario online, finalmente se usa un *support vector machine (SVM)* para la clasificación, este método supera al de *bag-of-words (BOW)*.

El trabajo de [Peng et al., 2008] fue publicado antes que el de [Samanta et al., 2012], pero a diferencia de éste, busca identificar la pose y orientación del bailarín en imágenes binarias. Los autores esperan que su trabajo tenga efectos en el desarrollo de ambientes interactivos de desempeño de danza basados en movimiento.

El método de [Kapsouras et al., 2013] usa el modelo de *BOW* para la clasificación de videos de danzas Griegas. Lo interesante de este trabajo es que se hace uso de 3 métodos generales de *activity recognition* para la extracción de características. El mejor resultado (78.68% de predicciones correctas) lo obtuvieron usando *Independent Subspace Analysis* [Le et al., 2011].

El trabajo de [Samanta and Chanda, 2013] proponen un método para la detección y caracterización mediante *space time interest point (STIP)* usando de geometría diferencial. Cada video es representado por un descriptor STIP para cada *frame*, la clasificación se hace mediante un SVM con x^2 kernel. Este método se uso para clasificar videos de 7 danzas consiguiendo una tasa de acierto de 68.18%.

Continuando sus trabajos previos[2], [Samanta and Chanda, 2014] presenta un nuevo enfoque para la clasificación de videos de danzas de India. Esta vez extrae propiedades para cada punto de interés como una matriz de covarianza que fusiona derivadas de espacio y tiempo. Obtiene una tasa de acierto de 69.39%.

El método de [Nussipbekov et al., 2014] busca reconocer los gestos de danzas

[2]Los trabajos previos son [Samanta and Chanda, 2013] y [Samanta et al., 2012]

típicas de Kazakh, este trabajo es especial porque usa el *Microsoft Kinect* para extraer datos (*depth images*). Se obtiene un tasa de acierto de 90.82% para 10 clases distintas de gestos.

El resumen de todos estos trabajos se muestra en la tabla 1.

Tabla 1: *Trabajos presentados en la Clasificación de Videos de Danzas Típicas.*

Nombre	Indian Classical Dance Classification by Learning Dance Pose Bases
Autor / Año	Soumitra Samanta, Pulak Purkait y Bhabatosh Chanda / 2012
Institución	Electronics and Communication Sciences Unit Indian Statistical Institute, Kolkata, India
Objetivo	Clasificar videos de danzas de la India
Conclusión	Se propone un nuevo descriptor que clasifica acciones humanas en general
Nombre	Binocular Dance Pose Recognition and Body Orientation Estimation via Multilinear Analysis
Autor / Año	Bo Peng y Gang Qian / 2008
Institución	Department of Electrical Engineering and Arts, Media & Engineering Program Arizona State University
Objetivo	Reconocer la posición del danzante y estimar de orientación
Conclusión	El análisis multilineal permite reconocer la posición y orientación
Nombre	Feature Comparison and Feature Fusion for Traditional Dances Recognition
Autor / Año	Ioannis Kapsouras, Stylianos Karanikolos, Nikolaos Nikolaidis y Anastasios Tefas / 2013
Institución	Department of Informatics, Aristotle University of Thessaloniki
Objetivo	Clasificar videos de danzas Griegas
Conclusión	Reconocer danzas es una tarea dicifil en la cual descriptores de *activity recognition* logran tasas de acierto de 8.68%
Nombre	A Novel Technique for Space-Time-Interest Point Detection and Description for Dance Video Classification
Autor / Año	Soumitra Samanta y Bhabatosh Chanda / 2013
Institución	Electronics and Communication Sciences Unit Indian Statistical Institute, Kolkata, India
Objetivo	Clasificar videos de danzas de la India
Conclusión	Se propone un nuevo método usado STIP y geometría diferencial
Nombre	Indian Classical Dance Classification on Manifold using Jensen-Bregman LogDet Divergence
Autor / Año	Soumitra Samanta y Bhabatosh Chanda / 2014
Institución	Electronics and Communication Sciences Unit Indian Statistical Institute, Kolkata, India
Objetivo	Clasificar videos de danzas de la India
Conclusión	Se propone un nuevo descriptor usando la forma y puntos de interés espacio temporales
Nombre	Kazakh Traditional Dance Gesture Recognition
Autor / Año	A K Nussipbekov, E N Amirgaliyev y Minsoo Hahn / 2014
Institución	Al-Farabi Kazakh National University, Kazakhstan, Korea Advanced Institute of Science and Technology, South Korea
Objetivo	Reconocer gestos de danzas de Kazakh
Conclusión	El uso del seguimiento de la cabeza del danzante mediante Microsoft Kinect mejora los resultados

1.1.2.2 Detección de Eventos Sociales

Social Event Detection (SED), Detección de Eventos Sociales en español, es un reto que busca establecer un *benchmark* para la detección de eventos sociales en datos multimedia que se encuentran en sitios web compartidos como Flickr, Picasa, Instagram y Twitpic — es parte del proyecto MediaEval.

Este reto busca que los participantes descubran y describan eventos sociales en la colección de datos, éstos incluyen imágenes y video. Se inició en 2011 y cuenta con 4 ediciones, en todos las ediciones el *dataset* incluye metadatos (etiquetas, ubicación, fecha):

- SED 2011: 73,645 imágenes, se divide en dos tareas:[Papadopoulos et al., 2011]
  - Encontrar los eventos de fútbol que se realizaron en Barcelona y Roma.
  - Encontrar los eventos que se realizaron en Paradiso(Holanda) y en Pare del Forum(España).
- SED 2012: 167,332 imágenes, se divide en tres tareas:[Papadopoulos et al., 2012]

  - Encontrar los eventos técnicos que ocurrieron en Alemania.
  - Encontrar los eventos de fútbol realizados en Hamburgo y Madrid.
  - Encontrar los eventos que muestren protestas realizadas en Madrid.

- SED 2013: 1,327 videos para la primera tarea y 27,754 imágenes para la segunda, éstas son:[Reuter et al., 2013]
  - Clusterizar el *dataset* de acuerdo a los eventos.
  - Determinar si una imagen muestra un evento o no, de ser así indicar su tipo.

- SED 2014: 362,578 imágenes para la primera tarea y 110,541 imágenes para la segunda, éstas son:[Petkos et al., 2014a]
  - Clusterizar el *dataset* de acuerdo a los eventos.
  - Dado el *dataset* recuperar aquellos eventos que cumplen con un criterio dado.

En las dos primeras ediciones los métodos propuestos hacen uso de recursos externos para extraer características y en base a éstos clasificar los eventos, en términos de *F-score* se obtuvieron resultados entre $67,95$ y $89,83$. En años posteriores se optó por operaciones de clusterización unimodal obteniendo un *F-score* de $0,946$. [Petkos et al., 2014b]

1.1.2.3 Reconocimiento de Eventos Culturales

Este problema se presenta como parte del IEEE *Conference on Computer Vision and Pattern Recognition*(CVPR) del 2015 con el nombre de Cultural Event Recognition. Nace a partir del trabajo de [Everingham et al., 2010] más conocido como *Action Classification Challenge of PASCAL VOC* y cuenta con 28,705 imágenes que fueron recolectadas de *Google Image* y *Bing Image* [Escalera et al., 2015][3], esto hace un total de 99 categorías que corresponden a eventos culturales de todo el mundo y 1 clase neutra. En todas las categorías la vestimenta, poses humanas, iluminación de objetos y el contexto constituyen posibles puntos para ser explotados en el reconocimiento del evento culturales [Escalera et al., 2015]. Se puede ver todas las categorías en la tabla 2.

[3]El dataset puede ser descargado de `https://www.codalab.org/competitions/4081`

Tabla 2: *Lista de las 100 categorías de eventos culturales*

Nro	Evento Cultural	País	#Imágenes	Nro	Evento Cultural	País	#Imágenes
1	Annual Buffalo Roundup	USA	230	51	4 de Julio	USA	301
2	Ati-atihan	Filipinas	221	52	AfrikaBurn	Sudáfrica	422
3	Balloon Fiesta	USA	270	53	Aomori Nebuta	Japón	401
4	Basler Fasnacht	Suiza	225	54	Apokries	Grecia	290
5	Boston Marathon	USA	236	55	Asakusa Samba Carnival	Japón	463
6	Bud Billiken	USA	261	56	Australia Day	Australia	241
7	Buenos Aires Tango Festival	Argentina	230	57	Bastille Day	Francia	232
8	Carnaval de Dunkerque	Francia	253	58	Beltane Fire	Escocia	361
9	Carnival of Venice	Italia	181	59	Boryeong Mud	Corea del Sur	300
10	Carnivale Rio	Brasil	282	60	Carnaval de Oruro	Bolivia	248
11	Casteller	España	322	61	Carnevale Di Viareggio	Italia	644
12	Chinese New Year	China	289	62	Cascamorras	España	223
13	Correfocs	España	251	63	Cheongdo Bullfighting Festival	Corea del Sur	212
14	Desert Festival of Jaisalmer	India	211	64	Crop Over	Barbados	262
15	Desfile de Silleteros	Colombia	226	65	Eid al-Adha	Egipto	282
16	Día de los Muertos	México	240	66	Eid al-Fitr Iraq	Iraq	282
17	Día da Sant Jordi	España	221	67	Epiphany	Grecia	270
18	Diwali Festival of Light	India	216	68	Festa Della Sensa	Italia	204
19	Falles	España	361	69	Frozen Dead Guy Festival	USA	230
20	Festa del Renaixement	España	263	70	Galugan	Indonesia	356
21	Festival de la Marinera	Perú	260	71	Grindelwald Snow Festival	Suiza	142
22	Inti Raymi	Perú	243	72	Hajj	Arabia Saudí	308
23	Fiesta de la Candelaria	Perú	243	73	Halloween Festival of Dead	USA	275
24	Gion Matsuri	Japón	258	74	Highland Games	Escocia	515
25	Harbin ice and snow Festival	China	276	75	JUnkanoo	Bahamas	290
26	Heiva	Tahití	236	76	Kaapse Klopse	Sudáfrica	229
27	Helsinki Samba Carnaval	Finlandia	229	77	Keene Pumpkin Festival	USA	275
28	Holi Festival	India	255	78	Krampusnacht Festival	Austria	142
29	Infiorata di Genzano	Italia	239	79	Los diablos danzantes	Venezuela	306
30	La tomatina	España	248	80	Magh Mela	India	200
31	Lewes Bonfire	Inglaterra	223	81	Mardi Gras	USA	333
32	Macys Thanksgiving	USA	235	82	Monkey Buffet Festival	Tailandia	188
33	Maslenitsa	Rusia	235	83	Naadam Festival	Mongolia	360
34	Midsommar	Suecia	259	84	Passover	Israel	273
35	Notting Hill Carnival	Inglaterra	244	85	Pflasterspektakel	Austria	218
36	Obon Festival	Japón	228	86	Phi Ta Khon	Tailandia	252
37	Oktoberfest	Alemania	329	87	Sahara Festival	Tunes	234
38	Onbashira Festival	Japón	228	88	Sapporo Snow Festival	Japón	227
39	Pingxi Lantern Festival	Taiwan	225	89	Spice Mas Carnival	Grenada	224
40	Pushkar Camel Festival	India	243	90	Sweden Medieval Week	Suecia	264
41	Quebec Winter Carnival	Canadá	244	91	Tamborrada	España	451
42	Queen's Day	Holanda	246	92	Tapati rapa Nui	Chile	244
43	Rath Yatra	India	264	93	Thaipusam	India	318
44	SandFest	USA	235	94	Thrissur Pooram	India	318
45	San Fermín	España	306	95	Tokushima Awa Odori Festival	Japón	354
46	Songkran Walter Festival	Tailandia	345	96	Tour de Francia	Francia	278
47	St Patrick's Day	Irlanda	248	97	Up Helly as Fire Festival	Escocia	224
48	The Battle of the Oranges	Italia	207	98	Vancouver Symphony on Fire	Escocia	224
49	Timkat	Etiopía	297	99	Vaisak Day	Indonesia	220
50	Viking Festival	Norway	241	100	Non-class	-	2000

Fuente: *[Escalera et al., 2015]*

Más de 50 trabajos se presentaron a esta competencia, el resumen de los métodos propuestos (10 mejores) se muestran en la tabla 3 y en un resumen de los resultados se muestran en la tabla 4.

Tabla 3: *Métodos presentados en el Cultural Event Recognition Challenge*

Equipo	Método Propuesto
VIPL-ICT-CAS	VGGNet, GoogleLeNet, Predicción usando Logistic Regression y LDA para la clasificación final.
FV	VGG16, VGG19, Place-CNN. Predicción usando 5 CNNs, fusión de 5 vectores de características y clasificación final con Logistic Regression.
MMLAB	GoogleLeNet, VGGNet, predicción usando activación de objetos y escenas. Características de CNN y Fisher Vector, clasificación final usando SVM.
NU&C	CaffeNet basada en ImageNet y Places205, combinación de CNN stream para objetos y CNN stream para escenas.
CVL.ETHZ	VGG-16 Basado en ImageNet y Places205, predicción usando Pooled y características LDA-projected CNN con clasificador basado en Iterative Neighbors.
SSTK	CNN, predicción usando la combinación de 13 CNN pre-entrenadas en place-206 e ImageNet.
MIPAL-SNU	GoogleNet, 4 redes entrenadas: región, imagen, persona, rostro y características CNN combinadas y predicción RF.
ESB	VGG16, GoogleNet, Predicción usando RF.
Sungbin Choi	GoogleNet basado en MIRFLICKR-1M e ImageNet. Promedio final para clasificación.
UPC-STP	AlexNet, SVM para la clasificación.

Fuente: *[Escalera et al., 2015]*

Tabla 4: *Resultado del Cultural Event Recognition Challenge*

Posición	Equipo	Desarrollo	Pruebas
1	VIPL-ICT-CAS	0.783	0.854
2	FV	0.770	0.851
3	MMLAB	0.717	0.847
4	NU&C	0.387	0.824
5	CVL-ETHZ	0.662	0.798
6	SSTK	0.740	0.770
7	MIPAL-SNU	0.801	0.763
8	ESB	0.729	0.758
9	SunBin Choi	-	0.624
10	UPC-STP	0.503	0.588

Fuente: *[Escalera et al., 2015]*

A diferencia de este problema, en la clasificación de imágenes de danzas típicas no se puede considerar la iluminación y contexto de la imagen como puntos a ser explotados ya que el danzante es el foco para la clasificación.

1.1.3 Justificación

Cusco es el principal destino turístico del país: a cada hora del día, 320 turistas pisan suelo cusqueño [4]. Esta gran cantidad de personas se combina con la basta diversidad cultural de la región para generar una necesidad de información sobre lo

[4]Fuente: diario La República `https://goo.gl/Oumg05`

que las personas ven a su alrededor. En el caso de las danzas, la variedad es tal que incluso para los propios cusqueños es difícil conocerlas todas.

A ésto se acopla la facilidad con la que actualmente se toman fotografías, generando mucha información visual que no es procesada ya que no existen trabajos para la clasificación de imágenes de danzas en la región.

1.1.4 Objetivos

1.1.4.1 Objetivo General

Aplicar técnicas de *machine learning* para la clasificación de imágenes de danzas típicas de Cusco.

1.1.4.2 Objetivos Específicos

Dentro de los objetivos específicos se incluyen:

- Explorar los métodos de extracción de características que permitan tener la información relevante de cada imagen.
- Crear un colección de imágenes de las danzas que son de interés para el trabajo.
- Explorar algoritmos de clasificación para evaluar y comparar los efectos que generan.
- Extraer puntos de interés, color, textura y bordes, así como comparar y analizar los resultados que generan en la detección y clasificación.

1.1.5 Evaluación

Para la evaluación del desempeño del trabajo se hará uso de la técnica de ***Cross Validation***, este método divide el conjunto de datos en 2 subconjuntos, escoge uno como elemento de prueba y el otro para el entrenamiento. Los indicadores principal son la tasa de acierto y tasa de error que son calculadas mediante:

$$Tasa\ de\ error = \frac{\#\ de\ predicciones\ incorrectas}{\#\ de\ elementos\ de\ prueba} \tag{1}$$

$$Tasa\ de\ acierto = 1 - Tasa\ de\ error \tag{2}$$

Se hará uso de ***Cross Validation*** ya que ha diferencia de métodos más exhaustivos como ***Leave-p-out cross-validation*** el costo computacional no crece exponencialmente y es además, un método bastante aceptado por la comunidad de *Computer*

Vision. Como indicadores secundarios de evaluación se hará uso de Precisión y Exactitud definidos por:

$$Recall = \frac{|Imágenes\ Relevantes \cap Imágenes\ Predichas|}{|Imágenes\ Relevantes|} \tag{3}$$

$$Presicion = \frac{|Imágenes\ Relevantes \cap Imágenes\ Predichas|}{|Imágenes\ Predichas|} \tag{4}$$

1.1.6 Alcances y Limitaciones

Debido a la complejidad del problema de clasificación de imágenes y los pocos estudios en este problema en específico, se generan los siguientes alcances y limitaciones:

- Se limita el número de danzas de interés a 4 [5], éstas son: **Qhapaq Chuncho, Qhapaq Qolla, Negrillos y Contradanza** [6] (figura 5).
- Actualmente no existe un *dataset* con imágenes etiquetadas de danzas de la región, por lo que fue necesario su creación y etiquetado.
- Para el procesamiento de los datos se usó una PC de escritorio común ya que no se cuenta con un servidor adecuado para este tipo de trabajos.
- Las técnicas que se usaron son en su mayoría de aprendizaje supervisado, ésto no implica que el uso de aprendizaje no supervisado se haya obviado.
- La clasificación dentro de la tasa de acierto mostrada en los resultados se limita a imágenes que cumplen con las características del *dataset* creado, ésto no impide que imágenes que no cumplen estas características puedan ser clasificadas de manera satisfactoria por el método propuesto.

[5] Se limita debido a la dificultad que presenta la recolección de datos: las fiestas no se realizan muy a menudo y por su naturaleza (cantidad de personas, cada comparsa está en lugares diferentes) la toma de fotografías se complica. Si bien se pueden recolectar imágenes en la web, la cantidad disponible no es suficiente para el procesamiento y no todas cumplen con las características que se desean.

[6] Se eligieron estas danzas porque existen similitudes en los colores de los atuendos en las parte superior (camisas blancas), la figura humana no es fuertemente distorsionada por los atuendos y las diferencias entre las máscaras de Qhapaq Chuncho y Contradanza son pequeñas.

Figura 5: *Danzas que son usadas en el trabajo*

(a) Danza Qhapaq Chuncho

(b) Danza Qhapaq Qolla

(c) Danza Negrillos

(d) Danza Contradanza

1.1.7 Metodología

Por la naturaleza del problema, el presente trabajo utiliza la metodología práctica[7]. Además, se utilizan las metodologías explorativa y descriptiva[8] para el análisis de los factores relevantes y desarrollo de la solución al problema.

El conjunto de fases que se siguen es:

1. **Revisión de literatura:** Se realiza un estudio de trabajos previos relacionados, la teoría de *machine learning* y procesamiento de imágenes. Ésto permite que las siguientes etapas sean posible.

2. **Recolección de imágenes:** En esta fase es necesario tomar fotografías de las danzas que son de interés. Los eventos principales para la recolección son las Fiestas del Cusco, Corpus Christi y Virgen del Carmen. Se debe resaltar que es necesario contar con imágenes desde ángulos distintos y en diferentes condiciones de iluminación. Al finalizar esta fase se debe contar con el *dataset* de las danzas de interés.

[7]Dentro de la clasificación presentada por[Mejía, 2005].
[8]Dentro de la clasificación presentada por[Hernández Sampieri et al., 2010].

3. **Investigación de Soluciones:** En esta etapa se busca posibles soluciones en base a la información de la fase 1. Al finalizar esta fase se debe contar con al menos 1 método de solución al problema.

4. **Implementación:** Ésta es la fase de desarrollo, se debe implementar los métodos de solución de la fase anterior.

5. **Evaluación y Análisis de resultados:** En base a todos los métodos desarrollados se debe evaluar la tasa de error de cada uno de ellos. En caso que ningun método alcance una tasa de error menor al 20%, se rechaza la hipotesis planteada; caso contrario, se acepta. Posteriormente se hace el análisis de los resultados y de los factores de mayor impacto.

1.1.8 Contribuciones

El presente trabajo tiene las siguientes contribuciones para la comunidad científica de visión computacional:

- Realiza un estudio sobre el desempeño de los métodos de *machine learning* en el problema de clasificación de imágenes de danzas típicas.

- Se creó un *dataset* de imágenes de danzas típicas que podrá ser usado de manera abierta por la comunidad científica.

Además el trabajo tiene el siguiente impacto social:

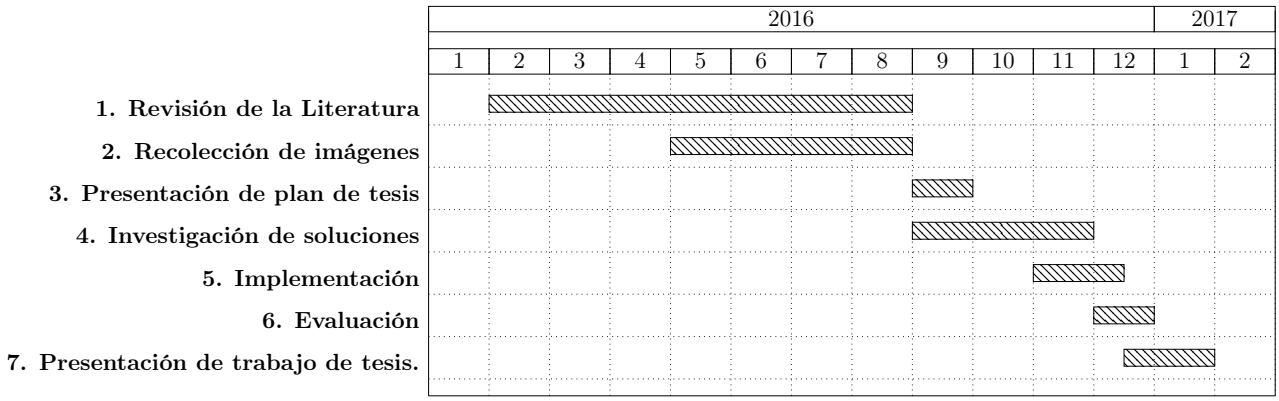
- Fomenta el desarrollo de trabajos de investigación de carácter tecnológico que tengan como base la cultura del Cusco y el Perú.

- Pone a disponibilidad de la sociedad un método que le permite conocer mejor su cultura.

1.1.9 Cronograma de Actividades

El cronograma de actividades se muestra en la figura 6.

Figura 6: *Diagrama de Gantt del cronograma de actividades.*



Capítulo 2

Marco Teórico

2.1 Procesamiento de Imágenes

En esta sección se muestra los conceptos básicos de *Procesamiento de Imágenes* que fueron necesarios para el desarrollo del proyecto.

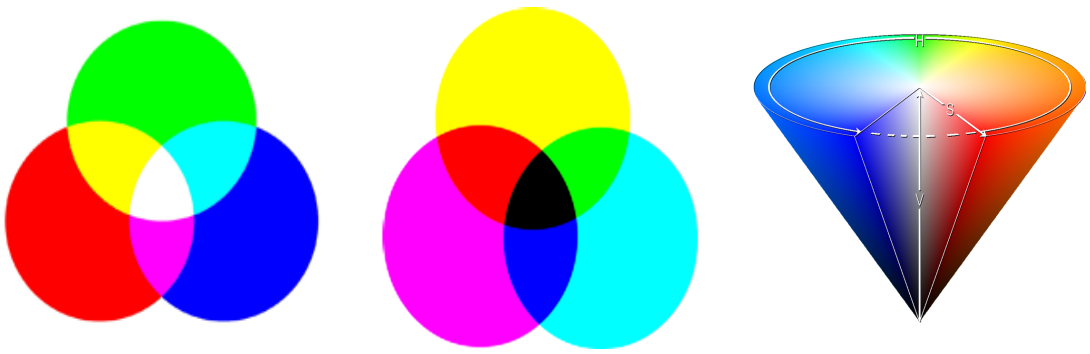
2.1.1 Colores

Existen diferentes sistemas para la representación de colores, los más conocidos son:

- **RGB(Red - Green - Blue)**. Este modelo combina los colores rojo, verde y azul para producir una gama de colores, es conocido también como el modelo aditivo.
- **HSV(Hue, Saturation, Value)**. Es una representación del modelo RGB pero en coordenadas cilíndricas. Como se observa en la figura 7 el canal H o matiz comúnmente indica el ángulo que corresponde a la base circular, mientras que los canales S y V corresponden al eje horizontal y vertical respectivamente.
- **CMYK(Cyan, Magenta, Yellow, Key)**. Este modelo es usado en las máquinas de impresión y al igual que el sistema RGB es un modelo aditivo.
- **Escala de Grises**. En este modelo cada píxel indica la intensidad de color.
- **Blanco y Negro**. También conocido como modelo binario, cada píxel almacena información que indica si debe ser blanco o negro.

[de los Santos Y., 2010]

Figura 7: *Sistemas de colores*



(a) RGB **(b)** CMYK **(c)** HSV

Fuente: *https://goo.gl/fBDs1p, https://goo.gl/4lc5Pv, https://goo.gl/nYSP3L*

2.1.2 Edge Detection

La detección de bordes es uno de los métodos de preprocesamiento en imágenes más importantes, usado para localizar los cambios en la función de intensidad. Los bordes son píxeles donde la función en mención (brillo) cambia abruptamente. Estudios en neurología y psico-física sugieren que en una imagen donde los cambios de la función de intensidad tienen cambios radicales son importantes para la percepción de la imagen. Si únicamente los lados de mayor magnitud son considerados, dicha información es suficiente para tener un entendimiento de la imagen. Gracias a ello, se puede reducir significativamente la información de la imagen, como se observa en la figura 8.[Torres, 2014]

Figura 8: *De izquierda a derecha. Imagen original, imagen con lados detectados.*

Fuente: *https://goo.gl/9zWZLn*

2.1.2.1 Canny Edge Detections

También conocido como el detector óptimo, fue presentado en el trabajo de [Canny, 1986] y busca satisfacer tres criterios: Minimizar la tasa de error, buena localización (la distancia entre los bordes detectados y los reales debe ser mínima) y respuesta mínima (una única respuesta por borde).

El algoritmo tiene cinco etapas, en la primera se aplica Gaussian Filter para eliminar el ruido de la imagen, en la segunda etapa se extrae el gradiente de intensidad, para ésto se aplica dos máscaras de convolución (para las direcciones X y Y):

$$G_x = \begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix} \tag{5}$$

$$G_y = \begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix} \tag{6}$$

El valor y dirección del gradiente se calcula mediante:

$$G = \sqrt{G_x^2 + G_y^2} \tag{7}$$

$$\theta = \arctan(\frac{G_x}{G_y}) \tag{8}$$

En la tercera etapa se aplica *Non-Maximun Suppression* para eliminar píxeles que no se consideran bordes. Se explicará esta técnica a detalle en la sección 2.1.8

La última etapa es denominada *Hysteresis*: se define un límite inferior y un límite superior. Si un píxel es menor al límite inferior se descarta, si es mayor al límite superior se acepta y si está entre ambos límites se acepta si está conectado a algún píxel que está sobre el límite superior. [Bradski, 2000]

2.1.3 Filtros

Los filtros son usados para suprimir las altas frecuencias, por ejemplo: para suavizar una imagen; o para bajas frecuencias, por ejemplo: para mejorar o detectar bordes. En esta sección se mostrará los filtros que fueron usados para el desarrollo del proyecto.

2.1.3.1 Gaussian

Es un operador 2D convolucional, es usado para suavizar una imagen y así eliminar detalles y ruido. La distribución Gaussiana en 1D está dada por:

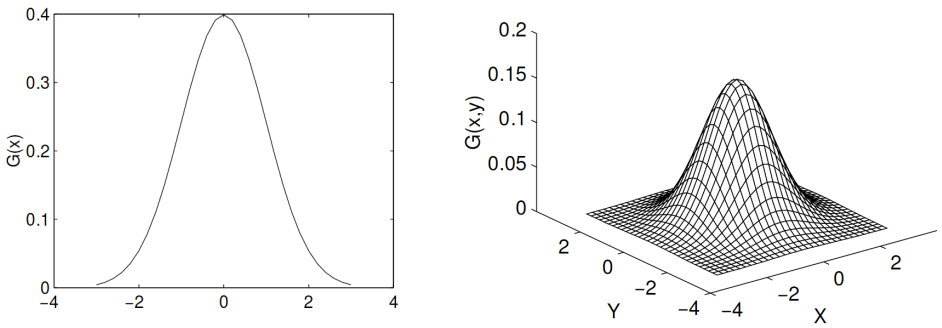
$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{9}$$

Donde σ es la desviación estándar de la distribución, se asume que la media es 0. Luego, una forma Gaussiana Isotrópica (circularmente simétrica) en 2D tiene la forma:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{10}$$

La idea del *Gaussian Filter* es usar esta distribución como una diferencia entre puntos, ésto se logra mediante la convolución. Dado que la imagen es una colección discreta de píxeles, es necesario generar una aproximación discreta de la función Gaussiana antes de realizar la convolución. Se puede ver el gráfico de las funciones en la figura 9.

Figura 9: *Gráficos de las ecuaciones 9 y 10.*



(a) Ecuación 9 **(b)** Ecuación 10

Fuente: *[Fisher et al., 1994]*

El efecto de este filtro es difuminar una imagen, el grado de suavizado depende de la desviación estándar de la función. Una forma fácil de comprender como trabaja el *Gaussian Filter* es verlo como el promedio ponderado de la vecindad de cada píxel, los píxeles más cercanos tiene mayor ponderación que los píxeles más alejados [Fisher et al., 1994]. La figura 10 muestra el efecto del *Gaussian Filter* sobre una imagen.

Figura 10: *De izquierda a derecha. Imagen original e imagen con Gaussian Filter.*

Fuente: *https://goo.gl/ml3uBg*

2.1.3.2 Bilateral

Este filtro suaviza las imágenes mientras preserva los bordes, para ésto usa una combinación no lineal de valores cercanos de la imagen. Una característica interesante de este filtro es que no produce colores fantasma alrededor de los bordes y también reduce los colores fantasma que aparecen en la imagen original [Tomasi and Manduchi, 1998].

El filtro se define mediante:

$$I^{filtered}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) F_r(|I(x_i) - I(x)|) g_s(|x_i - x|) \tag{11}$$

Donde el término de normalización:

$$W_p = \sum_{x_i \in \Omega} F_r(|I(x_i) - I(x)|) g_s(|x_i - x|) \tag{12}$$

Asegura que el filtro preserve la energía de la imagen, además se tiene:

- $I^{filtered}$ es la imagen con el filtro.
- I es la imagen original.
- x son las coordenadas del píxel que es filtrado.
- Ω es la ventana centrada en x.
- F_r es el kernel de rango que se usará para suavizar las diferencias de intensidad, ésta puede ser una función Gaussiana.
- g_s es el kernel de espacio para suavizar diferencias en coordenadas.

W_p es asignado usando las diferencias de distancia e intensidad. Considérese un píxel ubicado en (i, j) al cual se le quitara el ruido usando los píxeles (k, l) de su vecindad. Entonces la ponderación para el píxel (k, l) está dada por:

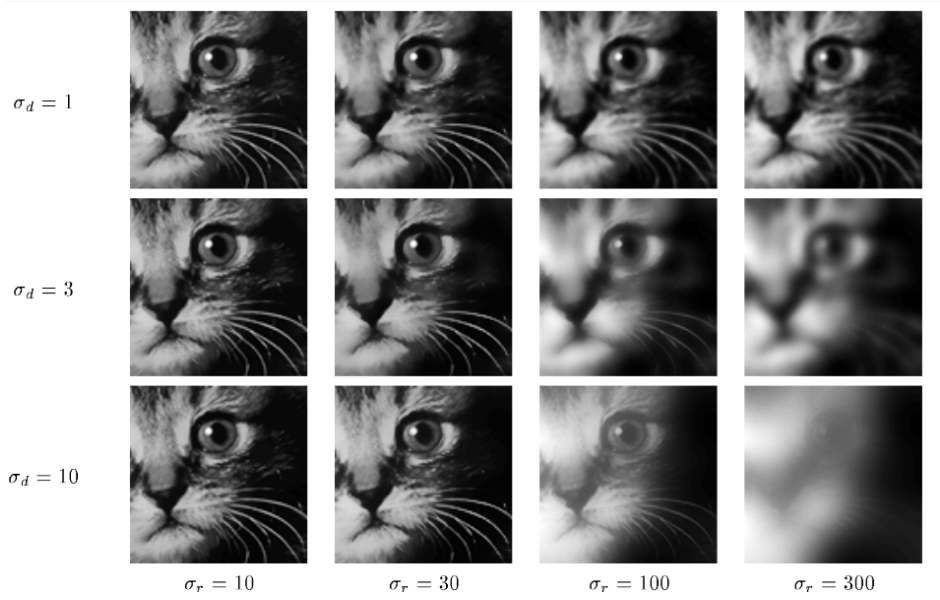
$$w(i, j, k, l) = e^{-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2} - \frac{|I(i,j)-I(k,l)|^2}{2\sigma_r^2}} \tag{13}$$

Donde σ_d y σ_r son los parámetros de suavizado, $I(i, j)$ y $I(k, l)$ son las intensidades de los píxeles (i, j) y (k, l) respectivamente. Después de calcular sus ponderaciones se normaliza mediante:

$$I_D(i, j) = \frac{\sum_{k,l} I(k, l) * w(i, j, k, l)}{\sum_{k,l} w(i, j, k, l)} \tag{14}$$

Donde I_D es la intensidad sin ruido del píxel (i, j). En la figura 11 se observa las variaciones de los resultados en base a los parámetros σ_d y σ_r. [Paris et al., 2009]

Figura 11: *Resultados de aplicar Bilateral Filter*



Fuente: *[Tomasi and Manduchi, 1998]*

2.1.4 Thresholding

En muchas aplicaciones, los niveles en escala de grises de los píxeles pertenecientes a objetos son sustancialmente diferentes a los niveles de los píxeles que forman parte del *background*. *Thresholding* representa una forma simple pero efectiva de separar los objetos del *background*.

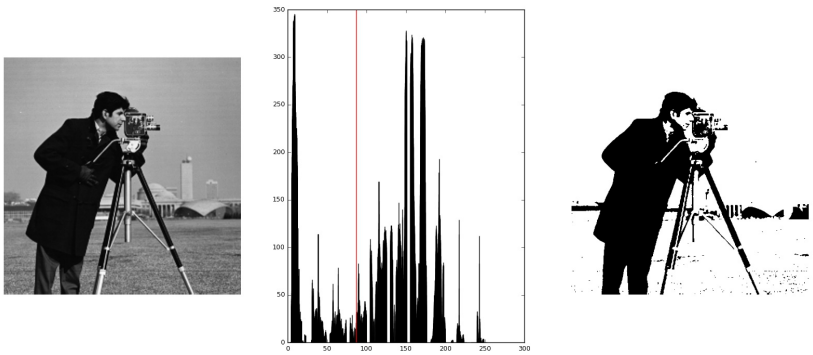
La salida generada por el proceso involucrado con el *thresholding* es una imagen binaria cuyo primer componente indica el *foreground* mientras que su complemento indica el *background*. Dependiendo de la aplicación, el *foreground* puede ser representado como el valor 0 en escala de grises (color negro) o el valor 255 (color blanco) [Torres, 2014].

En esta sección se muestra las dos maneras más generales de *thresholding*.

2.1.4.1 Simple Thresholding

En este tipo de *thresholding* se define un umbral, si la intensidad de un píxel es menor al umbral toma un valor 0, caso contrario 255; ésto puede variar según la definición de cada clase. Dentro de este tipo de *thresholding* se encuentra el método planteado por [Otsu, 1975], éste busca encontrar el mejor umbral maximizando la varianza de las dos clases. En la figura 12 se ve el mejor umbral definido por el método Otsu.

Figura 12: *De izquierda a derecha. Imagen original, histograma de la imagen junto al umbral del método Otsu (línea roja), imagen después del thresholding.*



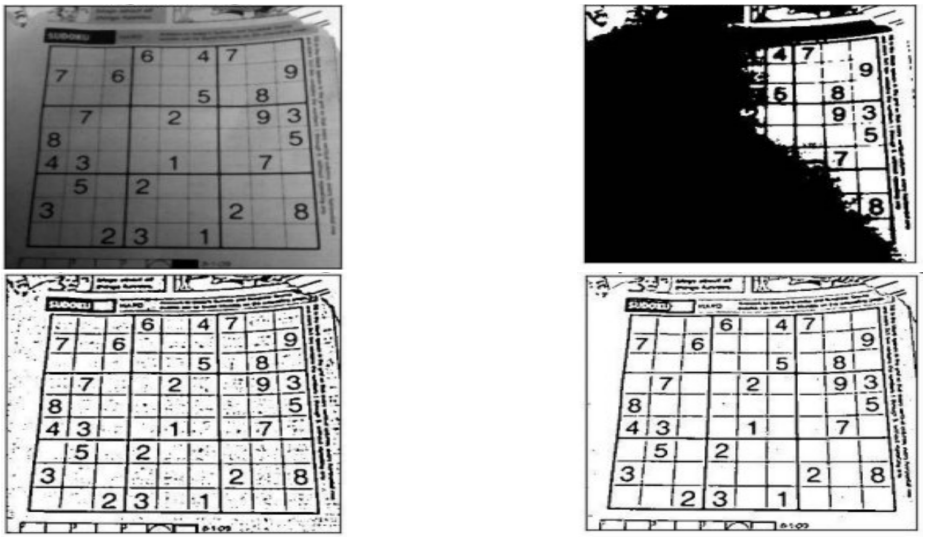
Fuente: *[Van der Walt et al., 2014]*

2.1.4.2 Adaptative Thresholding

Cuando se tienen imágenes con sombras, usar un valor global para el *thresholding* no da buenos resultados. Para estas situaciones se usa el *adaptative thresholding*.

Este tipo de *thresholding* calcula el umbral en una pequeña región de la imagen, permitiendo tener mejores resultados en situaciones de cambio de iluminación. La forma en la que se calcula el umbral es usando el promedio o una función Gaussiana. En la figura 13 se observa los resultados al realizar *thresholding* con los diferentes métodos.

Figura 13: *De izquierda a derecha, de arriba hacia abajo. Imagen original, simple thresholding (umbral = 127), adaptative mean thresholding, adaptative gaussian thresholding.*



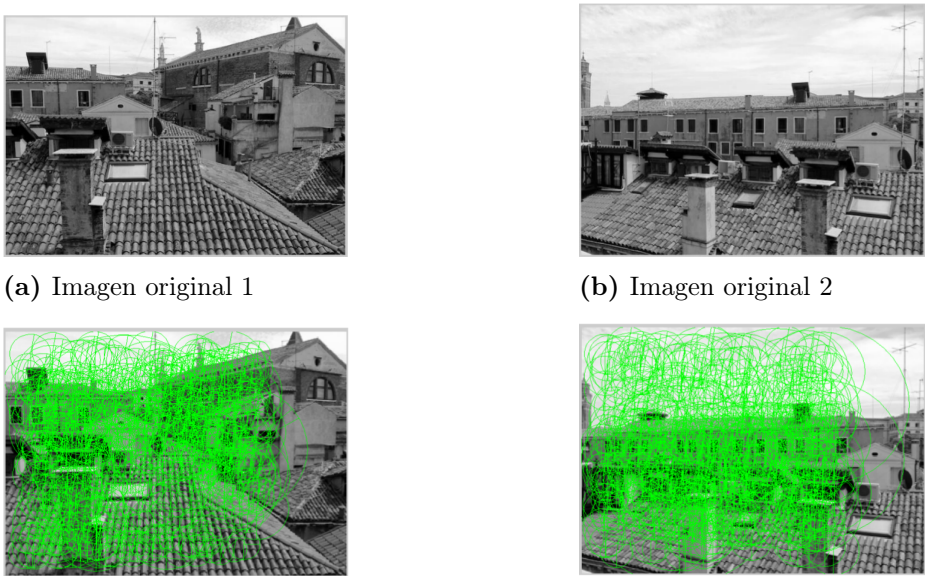
Fuente: *[Bradski, 2000]*

2.1.5 Descriptores

2.1.5.1 Scale-invariant feature transform (SIFT)

Este método fue presentado por [Lowe, 2004] y busca extraer características que sean invariantes a la rotación y escalamiento, y parcialmente invariables a cambios de iluminación y puntos de vista. Una característica importante es que método es que genera una gran cantidad de características que cubren toda la imagen. Además, éstas son altamente diferenciales, lo cual le permite se tenga mejores resultados al hacer una comparación entre imágenes. En la figura 14 se vé SIFT aplicado a dos imágenes del mismo lugar.

Figura 14: *Extracción de SIFT en dos imágenes del mismo lugar pero de ángulos distintos*



(a) Imagen original 1

(b) Imagen original 2

(c) SURF en la imagen original 1

(d) SURF en la imagen original 2

Fuente: *[Panchal et al., 2013]*

Este método cuenta con 4 etapas principales:

- El primer paso consiste en buscar sobre todas las escalas y ubicaciones de la imagen. Para hacer ésto de manera eficiente se hace uso de una función de diferencias Gaussianas para identificar puntos de interés potenciales que son invariantes a la escala y orientación.
- Para cada punto de interés candidato se desarrolla un modelo detallado para determinar posición y escala. Se selecciona a aquellos que muestren mayor estabilidad.
- Se asigna 1 o más orientaciones para cada punto de interés basado en el gradiente de la dirección de la imagen.
- Se calcula el gradiente en la escala escogida alrededor de los puntos de interés. Éstos se transforman en una representación que admite un nivel significativo de distorsión y cambio de iluminación.

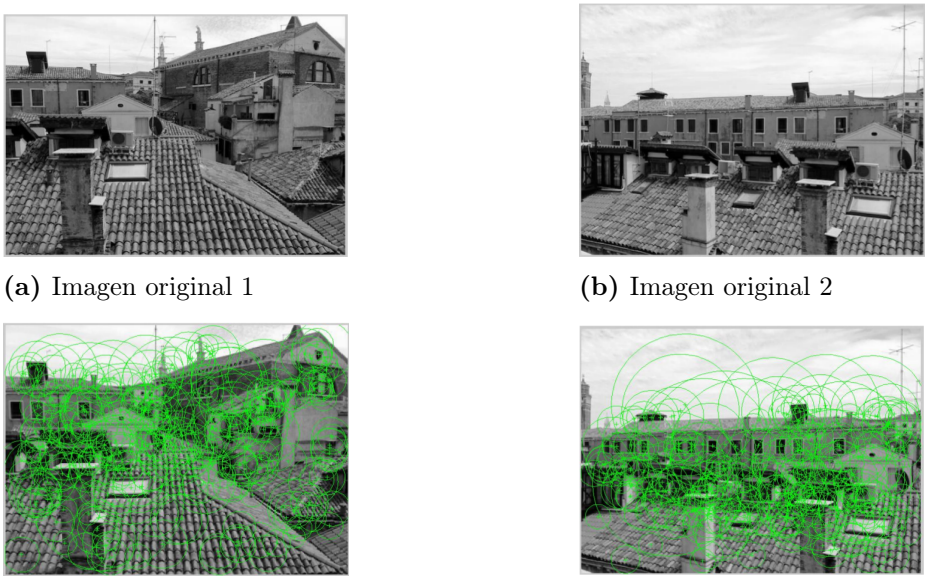
2.1.5.2 Speeded up robust features (SURF)

Fue planteado por [Bay et al., 2006] y al igual que SIFT busca ser invariante a la rotación y escalamiento. Consta de 2 etapas principales:

- Se selecciona los puntos de interés en locaciones distintivas (*corners, bobs, T-juctions*).
- Se representa la vecindad de cada punto de interés con un vector de características. Éstos deben ser distintivos y al mismo tiempo robustos al ruido, errores de detección y deformaciones geométricas y fotométricas.

Como se puede ver en el trabajo de [Panchal et al., 2013] SIFT permite obtener mayor cantidad de puntos de interés que SURF, pero es lento. SURF al igual que SIFT muestra un buen desempeño. En la figura 15 se vé SURF aplicado a dos imágenes de mismo lugar.

Figura 15: *Extracción de SURF en dos imágenes del mismo lugar pero de ángulos distintos*



(a) Imagen original 1 **(b)** Imagen original 2

(c) SURF en la original 1 **(d)** SURF en la original 2

Fuente: *[Panchal et al., 2013]*

Existen modificaciones al algoritmo original, entre ellas *Upright SURF* o *U-SURF* que a diferencia del método clásico no es invariante a la rotación pero es mucho más rápido y tiene mejor desempeño para aplicaciones en las que las imágenes son en su mayoría horizontales o verticales.

2.1.5.3 Local Binary Pattern (LBP)

Este es un descriptor de textura que se volvió popular por el trabajo de [Ojala et al., 2002], ha demostrado ser invariante a las rotaciones.

A diferencia de otros descriptores como el *Haralick texture feature* [Haralick et al., 1973] que calcula la representación global en base a la matriz de co-ocurrencia de escala de

grises, LBP calcula una representación local.

Esta representación local se construye combinando los valores de la vecindad de un píxel:

$$I^{loc}_{(k,l)} = I_{(i,j)} > I_{(k,l)} ? 1 : 0 \tag{15}$$

Donde (i, j) el píxel central, (k, l) es el píxel de la vecindad de (i, j), $I_{(i,j)}$ y $I_{(k,l)}$ son los valores en la imagen original y I^{loc} es la matriz de representación local. En base los valores de esta matriz se calcula el valor final en la imagen LBP para el píxel (i, j) mediante:

$$I^{lbp}_{(i,j)} = \sum_{i=0}^{\sum_{j=0}^{j<r} 2^{j+3}} 2^i I^{loc}_{(k,l)} \tag{16}$$

Donde r indica tamaño de la vecindad y $I^{lbp}_{(i,j)}$ es el valor LBP del píxel (i, j).

Considere el siguiente ejemplo que es una pedazo de una imagen con $r = 1$, el centro de la matriz es el píxel para el cual se calculará su valor en la imagen LBP.

$$\begin{vmatrix} 8 & 6 & 3 \\ 6 & \mathbf{4} & 2 \\ 1 & 8 & 1 \end{vmatrix}$$

Usando la ecuación 15, la matriz de representación local es:

$$\begin{vmatrix} 0 & 0 & 1 \\ 0 & & 1 \\ 1 & 0 & 1 \end{vmatrix}$$

Mediante la ecuación 16 y siguiendo el orden definido en la siguiente matriz se puede calcular el valor en la imagen LBP para el píxel central.

$$\begin{vmatrix} 6 & 7 & 0 \\ 5 & & 1 \\ 4 & 3 & 2 \end{vmatrix}$$

Ésto es: $0 * 2^7 + 0 * 2^6 + 0 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 23$, en la figura 16 se observa el resultado de aplicar LBP a una imagen.

Figura 16: *De izquierda a derecha. Imagen original, imagen después de LBP e histograma de imagen LBP.*

Fuente: `https://goo.gl/yvgGC7`

Si bien las ecuaciones anteriores definen la imagen *LBP*, ésta no es directamente usada como descriptor. Se usa el histograma que se genera a partir de la imagen *LBP* para este fin. [Van der Walt et al., 2014]

2.1.6 Image Entropy

La entropía en una imagen, es la cantidad de información que contiene. Imágenes con poca entropía, como una que contiene en su mayoría un cielo azul, tienen poco contraste y muchos píxeles con el mismo o valores similares, ésto hace que esta imagen pueda ser comprimida en poco espacio. Por otro lado, una imagen con mucha entropía, como una que tiene la vestimenta de una danza colorida con cambios de iluminación, tiene un gran problema de contraste por lo que no puede ser comprimida en el mismo espacio que la imagen con poca entropía. La entropía se define mediante:

$$Entropy = -\sum_{i} P_i Log_2 P_i \tag{17}$$

Donde P_i es la probabilidad de que la diferencia entre dos píxeles adyacentes sea igual a i [O'Brien, 1997]. El concepto de entropía es también usado dentro del procesamiento del lenguaje natural para referirse al concepto matemático usado en teoría de la información, también conocida como entropía de Shannon.

2.1.7 Hough Transform

Hough Transform es una técnica que busca detectar instancias imperfectas de objetos con una determinada forma. Inicialmente fue planteado para la detección de líneas, posteriormente se extendió la idea para detectar formas más complejas como círculos y elipses.

Lo interesante de esta técnica es que al detectar una forma, no espera que ésta sea perfecta. En la mayoría de casos existen píxeles faltantes, pero ésto no hace que

Hough Transform no pueda estimar de manera correcta la forma.

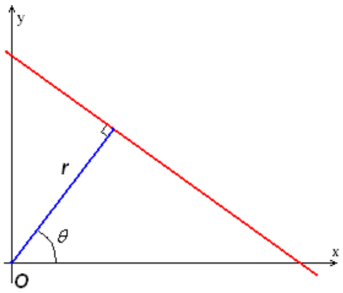
Una línea que sigue la ecuación:

$$Y = mX + b \tag{18}$$

Puede ser representada como el par (m, b), ésto permite ver que la línea puede ser representada por (r, θ) ya que existe la ecuación 19 que representa la figura 17.

$$r = X \cos\theta + Y \sin\theta \tag{19}$$

Figura 17: *Gráfico de la ecuación 19.*



Fuente: `https://goo.gl/zOFnDC`

Donde r es la distancia desde el origen al punto más cercano de la línea, y θ es el ángulo entre el eje x y la línea que pasa por el origen y el punto más cercano. El plano (r, θ) es conocido como el espacio *Hough*.

Para un punto en el plano, todas las líneas que pasan por este punto forman la curva sinusoidal en el plano (r, θ) que es único para ese punto. Un conjunto de dos o más puntos que forman una línea producirán sinusoidales que cruzan (r, θ) para esa línea. De esta manera el problema de detectar puntos colineales se puede transformar en un problema de encontrar curvas concurrentes.

Para transformar este algoritmo para detectar círculos se debe seguir los siguientes pasos:

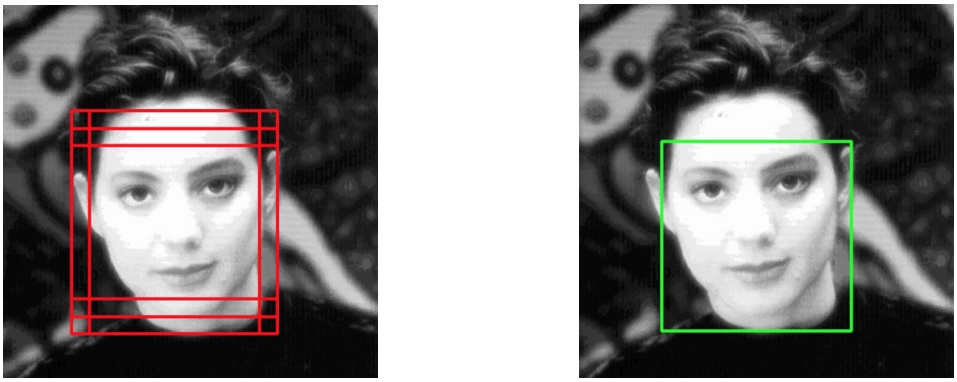
- Se crea un espacio de acumulación que está hecho por una celda para cada píxel. Se inicializa las celdas en 0.
- Para cada punto (i, j) en la imagen, incremente todas las celdas de acuerdo a la ecuación de un círculo: $(i - a)^2 + (j - b)^2 = r^2$, donde (a, b) es el centro y r es el radio del círculo, r es constante. Ésto es, para cada posible valor a calcule todos los posibles b que satisfacen la ecuación del círculo.
- Busque el máximo local en espacio de acumulación, éstos representan los círculos detectados por el algoritmo.

[Van der Walt et al., 2014]

2.1.8 Non-Maximum Suppression

Non-Maximum Suppression es un técnica que te permite eliminar elementos que se solapan dentro de una imagen. Ésto es muy útil cuando se tiene un método que detecta objetos y los resultados finales se solapan (figura 18).

Figura 18: *De izquierda a derecha. Resultados de un método de detección de rostros, resultados después de Non-Maximum Suppression.*



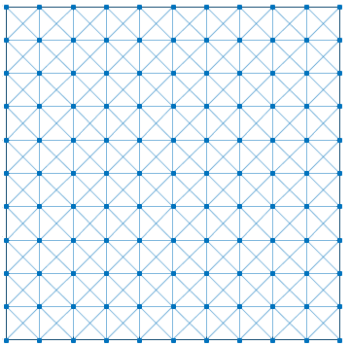
Fuente: *https://goo.gl/zOFnDC*

El algoritmo planteado inicialmente por [Girshick et al., 2010] fue mejorado por Tomasz Malisiewicz et al.quitando un bucle interno. La idea se resume en ordenar los resultados de la detección (*boxes*) por su componente y y recorrerlos de atrás hacia adelante: para cada box_i se calcula el porcentaje de solapamiento con $box_j, j < i$ y se elimina aquellos que tengan un porcentaje mayor a un umbral predefinido.

2.1.9 Recorrido en Grafo

Una forma de trabajar sobre imágenes es usar el grafo equivalente para el procesamiento. Para ésto se considera que cada píxel es un nodo y está conectado con los píxeles de su vecindad, una medida clásica es considerar los 8 píxeles que lo rodean (Figura 19). [Silvela and Portillo, 2001]

Figura 19: *Grafo formado a partir de una imagen.*



Fuente: `https://goo.gl/5dvlBv`

Con esa premisa, el uso del algoritmo de Búsqueda en Anchura (*Breadth-first search (BFS)*) y en Profundidad (*Depth-first search (DFS)*) es posible. Se muestra a continuación el pseudocódigo de la adaptación de BFS para imágenes.

```
1: function BFS (Image)
2:     Limpiar Lista_Visitados
3:     for pixel (i, j) ∈ Image do
4:         if noVisitado(i, j) then
5:             Limpiar Cola Q
6:             q.push((i, j))
7:             while not q.empty() do
8:                 current = q.front()
9:                 q.pop()
10:                Lista_Visitados.push(current)
11:                for pixel (k, l) adyacente a (i, j) do
12:                    if noVisitado(k, l) then
13:                        q.push((k, l))
14:                        Lista_Visitados.push((k, l))
15:                    end if
16:                end for
17:            end while
18:        end if
19:    end for
20: end function
```

Fuente: *[Silvela and Portillo, 2001]*

2.2 Machine Learning

El *Machine Learning* o Aprendizaje Automático se puede definir como un área de investigación que busca darle a las computadoras la habilidad de aprender sin ser explícitamente programadas [Samuel, 1959].

Si bien el rango de problemas en los cuales trabaja el *Machine Learning* es amplio, se le puede agrupar en:

- **Clasificación:** En este tipo de problemas se resume en dado un conjunto de datos o patrones determinar la clase a la cual pertenecen los datos. Por ejemplo, determinar si un tumor es maligno dado su tamaño y la edad de la persona. La forma más básica de clasificación es la binaria, en ésta se busca determinar si el conjunto de datos pertenecen a la clase de interés o no.

- **Regresión:** Estos problemas buscan determinar el valor real de una variable y dado el conjunto de datos o patrones x, es muy parecido a los problemas de clasificación con la diferencia que el rango de la función de predicción de continuo.

- **Clusterización:** En este tipo de problemas se busca formar clusters o grupos dentro de datos.

Los métodos de *Machine Learning* se pueden clasificar por la forma en la cual los datos son dados al algoritmo: *Supervised Learning* y *Unsupervised Learning*. Existen también otros tipos como *Reinforcement Learning* pero éstos no son usados en el presente trabajo.

El *Supervised Learning* o Aprendizaje Supervisado recibe datos y etiquetas sobre estos datos, estas etiquetas contienen la predicción correcta y es en base a éstas que el algoritmo construye un modelo de predicción. Este tipo de modelos son usados para problemas de clasificación y regresión.

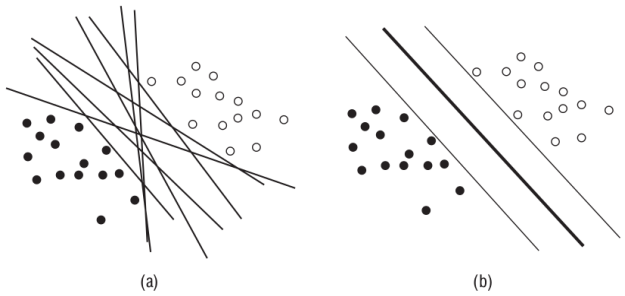
El *Unsupervised Learning* o Aprendizaje No Supervisado recibe datos mas no etiquetas. En base a la entrada el algoritmo busca construir un modelo que describa la estructura de los datos. Este tipo de algoritmos son usados para problemas de clusterización.

2.2.1 Support Vector Machine (SVM)

Es un modelo usado para separar dos clases distintas. La principal diferencia con un modelo lineal simple es que *SVM* busca optimizar la estructura que separa las

clases (líneas para dos dimensiones, planos para 3 dimensiones e hiperplanos para más dimensiones) [Parker, 2011], ésto se puede ver en la figura 20.

Figura 20: *SVM. Los círculos blancos y negros son las dos clases. (a) El conjunto de las posibles líneas que separan las dos clases, (b) La mejor línea que separa las clases.*



Fuente: *[Parker, 2011]*

Asúmase que se tiene L datos de entrada, cada dato x_i tiene D atributos y pertenecen a las clases $y = 1$ o $y = -1$, entonces el conjunto de datos de entrenamiento es:

$$\{x_i, y_i\} \; donde \; i = 1 \; ... \; L, \; y_i \in \{-1, 1\}, x \in \Re^D$$

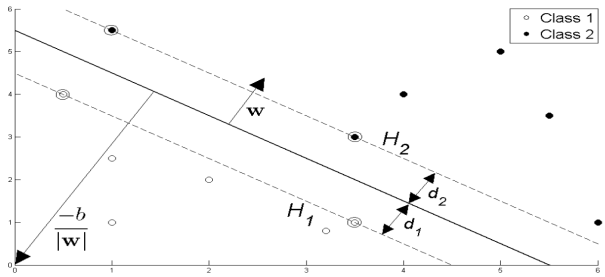
Se asume que los datos son linealmente separables, ésto quiere decir que se puede dibujar una línea en el gráfico de x_1 *vs* x_2 separando las dos clases cuando $D = 2$ y en el hiperplano de $x_1, x_2...x_D$ cuando $D > 2$. En casos en los que no sea posible esta separación lineal, se usa un *Gaussian Kernel* para la solución.

El hiperplano puede ser descrito como $w.x + b = 0$, donde

- w es la normal del hiperplano al origen.
- $\frac{b}{|w|}$ es la distancia perpendicular del hiperplano al origen.

Support Vectors son los puntos más cercanos a hiperplano de separación y el objetivo del *Support Vector Machine* están orientados a este hiperplano de manera tal que esté lo más alejado posible de los miembros más cercanos de ambas clases (figura 21).

Figura 21: *Hiperplanos en dos clases separables linealmente.*



Fuente: *[Fletcher, 2009]*

Considerando la figura 21 permite seleccionar los parámetros w y b de manera que el conjunto de entrenamiento puede ser descrito por:

$$x_i.w + b \geq +1 \; para \; y_i = +1 \tag{20}$$

$$x_i.w + b \leq -1 \; para \; y_i = -1 \tag{21}$$

Estas ecuaciones pueden ser combinadas en:

$$y_i(\; x_i.w + b) \;\; \geq 0 \; \forall i \tag{22}$$

Si ahora sólo se considera los puntos que están más cerca al hiperplano de separación, los hiperplanos H_1 y H_2 a los cuales pertenecen estos puntos se pueden describir como:

$$H_1 : \;\; x_i.w + b = +1 \tag{23}$$

$$H_2 : \;\; x_i.w + b = -1 \tag{24}$$

De la figura 21 se tiene que d_1 y d_2 son las distancias de H_1 y H_2 al hiperplano respectivamente. En un plano equidistante $d_1 = d_2$, esta cantidad es conocida como margen del SVM y es ésto lo que se busca maximizar.

Por geometría se ve que el margen es igual a $\frac{1}{|w|}$ y maximizar este valor sujeto a las condiciones de la ecuación 22 es equivalente a encontrar:

$$min \;\; |w| \;\; / \; y_i(\; x_i.w + b) \;\; - 1 \geq 0 \; \forall i \tag{25}$$

Minimizar $|w|$ es equivalente a minimizar $\frac{1}{2}|w|^2$ y el uso de este término permitirá realizar una optimización mediante Programación Cuadrática posteriormente.

Entonces se tiene:

$$min\ \frac{1}{2}\left|w\right|^2\ /\ y_i(\ x_i.w+b)\ -1\geq 0\ \forall i \tag{26}$$

A fin de cumplir las condiciones de esta ecuación es necesario introducir el multiplicador de Lagrange α, donde $\alpha \geq 0\ \forall i$:

$$L_P \equiv \frac{1}{2}\left|w\right|^2 - \alpha[\ y_i(\ x_i.w+b)\ -1\geq 0\ \forall i] \tag{27}$$

$$L_P \equiv \frac{1}{2}\left|w\right|^2 - \sum_{i=1}^{L}\alpha_i[\ y_i(\ x_i.w+b)\ -1] \tag{28}$$

$$L_P \equiv \frac{1}{2}\left|w\right|^2 - \sum_{i=1}^{L}\alpha_i y_i(\ x_i.w+b)\ +\sum_{i=1}^{L}\alpha_i \tag{29}$$

Entonces se desea encontrar w y b que minimicen, y α que maximicen la ecuación 29 (manteniendo $\alpha \geq 0\ \forall i$). Se puede hacer ésto mediante la diferencial de L_P con respecto a w y b y fijando la diferencial a:

$$\frac{\partial L_P}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{L}\alpha_i y_i x_i \tag{30}$$

$$\frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{i=1}^{L}\alpha_i y_i = 0 \tag{31}$$

Reemplazando las ecuaciones 30 y 31 en 29 da una nueva formulación que depende de α, se debe maximizar:

$$L_D \equiv \sum_{i=1}^{L} -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i.x_j\ /\ \alpha_i \geq 0\ \forall i, \sum_{i=1}^{L}\alpha_i y_i = 0 \tag{32}$$

$$L_D \equiv \sum_{i=1}^{L} -\frac{1}{2}\sum_{i,j}\alpha_i H_{i,j}\alpha_j\ donde\ H_{i,j} \equiv y_i y_j x_i.x_j \tag{33}$$

$$L_D \equiv \sum_{i=1}^{L} -\frac{1}{2}\alpha^T H\alpha\ /\ \alpha_i \geq 0\ \forall i, \sum_{i=1}^{L}\alpha_i y_i = 0 \tag{34}$$

La nueva fórmula para L_D es conocida como la forma *dual* de la forma primaria de L_P. Este es un problema de optimización cuadrática, se puede ejecutar un solver en la ecuación 30 que nos permitirá obtener w, ésto deja sólo b para calcular.

Cualquier punto que satisface la ecuación 31 que es un *Support Vector* x_s tendrá

la forma:

$$y_s(\ x_s.w + b)\ = 1 \tag{35}$$

Reemplazando en la ecuación 30:

$$y_s \left(\sum_{m \in S} \alpha_m y_m x_m . x_s + b \right) = 1 \tag{36}$$

Donde S denota el conjunto de índices de los *Support Vector*. S es determinado buscando los índices i donde $\alpha_i > 0$. Multiplicando por y_s y después usando $y_s^2 = 1$ de las ecuaciones 20 y 21:

$$y_s^2 \left(\sum_{m \in S} \alpha_m y_m x_m . x_s + b \right) = y_s \tag{37}$$

$$b = y_s - \sum_{m \in S} \alpha_m y_m x_m . x_s \tag{38}$$

En vez de considerar un *Support Vector* arbitrario es mejor obtener el promedio de todos los *Support Vector* en S:

$$b = \frac{1}{N_s} \sum_{s \in S} (\ \alpha_m y_m x_m . x_s) \tag{39}$$

Ahora ya se tiene las variables w y b que definen el hiperplano de separación y por lo tanto el *SVM* [Fletcher, 2009].

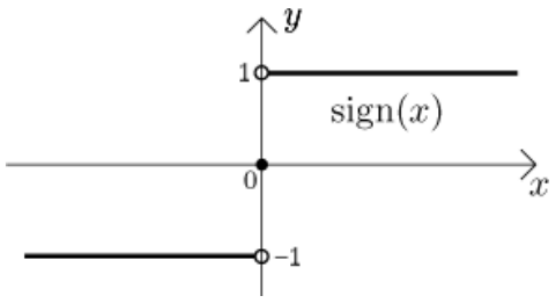
2.2.2 Adaboost

Este es un algoritmo busca construir un clasificador fuerte (*strong*) en base a la combinación lineal de clasificadores simples (*weak*) $h_t(x)$ (por ejemplo un árbol de decisión):

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{40}$$

Donde T es el número de clasificadores simples, $H(x) = sign(f(x))$ es el clasificador fuerte o hipótesis final y α_t es la ponderación de cada clasificador simple, *sign* es la función Signum (figura 22).

Figura 22: *Función Signum*



Fuente: *https://goo.gl/OLXrfD*

Sea un conjunto de m datos $(x_1, y_1), \ldots, (x_m, y_m)/x_i \in X, y_i \in \{-1, 1\}$ se calcula *Adaboost* de la siguiente manera:

- Se inicializa los pesos $D_1(i) = 1/m$
- Para $t = 1, \ldots, T$:
  1. Procese la salida del clasificador simple, $h_t : X \to \{-1, 1\}, H = \{h(x)\}$ con menor error de distribución D_t.
  2. Escoger $\alpha_t \in \mathbb{R}$
  3. Actualice:
  $$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t} \tag{41}$$
  Donde Z_t es un factor de normalización.
- El resultado final se obtiene de:
$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{42}$$

Para procesar el clasificador con el menor error:

$$h_t = arg \min_{h(x) \in H} \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)] \tag{43}$$

Debe cumplirse además: $\epsilon_t < 1/2$, caso contrario se detiene el bucle.
El objetivo es minimizar:

$$\varepsilon_{tr} = \frac{1}{m} |i : H(x_i) \neq y_i| \tag{44}$$

Ésto genera una cota superior:

$$\varepsilon_{tr}(H) \leq \prod_{t=1}^{T} Z_t \tag{45}$$

Ahora se debe explicar cómo se calcula α_t. Este valor se selecciona para minimizar $Z_t(\alpha)$ de manera *greedy*.

$Z_t(\alpha)$ es una función convexa diferenciable en un extremo, el α_t óptimo se obtiene mediante:

$$\alpha_t = \frac{1}{2} \log \left(\frac{1 + r_t}{1 - r_t} \right) \tag{46}$$

Donde:

$$r_t = \sum_{i=1}^{m} D_t(i) h_t(x_i) y_i \tag{47}$$

Para el óptimo α_t se cumple:

$$Z_t = 2\sqrt{\epsilon_t(1 - \epsilon_t)} \leq 1 \tag{48}$$

[Matas and Sochman, 2004]

2.2.3 Random Forest

Random Forest fue planteado por [Breiman, 2001] y es la combinación de varios árboles predictores de manera que cada árbol depende de los valores de un vector aleatorio independiente, que a la vez tiene la misma distribución para todos los árboles.

Sea los datos de entrenamiento un conjunto n de vectores $X_i = x_1, x_2, \ldots, x_n$ etiquetados con Y_i, $i \in 1, \ldots, n$. Para la construcción del *Random Forest* se crean k árboles, cada uno de estos árboles es entrenado con un subconjunto de X_i formado de manera aleatoria. Una vez que se tienen entrenados cada uno de estos árboles se usa el proceso de *bagging* para combinar sus resultados.

Bagging es el proceso por el cual se realiza una votación para las clases existentes, aquella con mayor votación es la clase que predice el *Random Forest*. A continuación, se explica porque este modelo es óptimo y logra converger.

Dado un conjunto de clasificadores $h_1(x), h_2(x), \ldots, h_K(x)$ con un conjunto de entrenamiento generado de manera aleatoria a partir de los vectores X y Y, se define la función margen:

$$mg(X, Y) = av_k I(h_k(X) = Y) - \max j \neq Y av_k I(h_k(X) = j) \tag{49}$$

Donde $I(.)$ es la función indicador. El margen indica la medida en que el número

medio de votos de X, Y para la clase correcta excede los votos para el resto de clases. Mientras más largo sea el margen, mayor la confianza de la clasificación. El error de generalización está dado por:

$$PE* = PX, Y(mg(X, Y) < 0) \tag{50}$$

Donde el subíndice X, Y indica que la probabilidad se define sobre el espacio de X, Y.

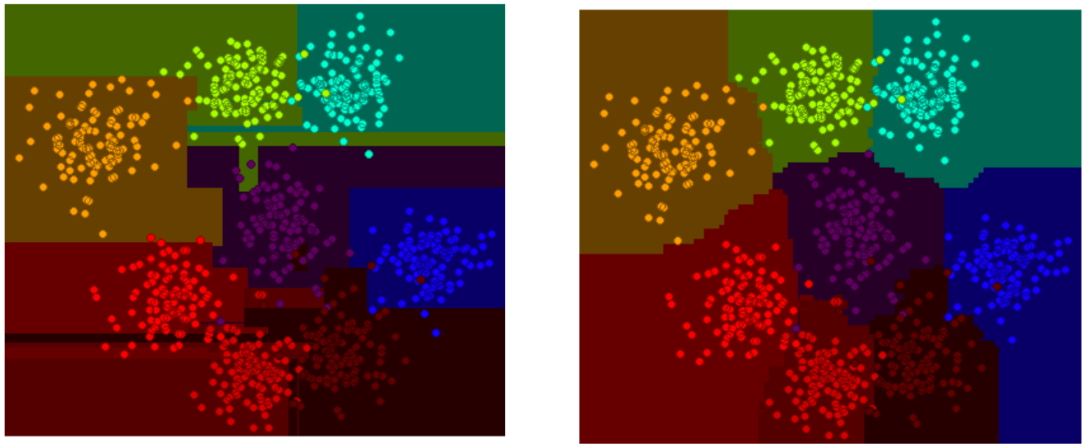
En los *Random Forest*, $h_k(X) = h(X, \Theta_k)$ donde $h(X, \Theta_k)$ es el clasificador entrenado con el vector aleatorio Θ_k. Para un número grande de árboles, se cumple por la ley fuerte de grandes números y la estructura de árbol: A medida que el número de árboles crece, para casi para todas las secuencias Θ, $PE*$ converge a:

$$P_{X,Y}(P_\Theta(h(X, \Theta) = Y) - \max_{j \neq Y} P_\Theta(h(X, \Theta) = j) < 0) \tag{51}$$

[Breiman, 2001]

En la figura 23 se muestran los diferentes resultados que se obtiene al variar el parámetro k (número de árboles).

Figura 23: *Resultados de Random Forest para diferentes valor de k.*



(a) k = 1 **(b)** k = 500

Fuente: *https://goo.gl/oqr9u1*

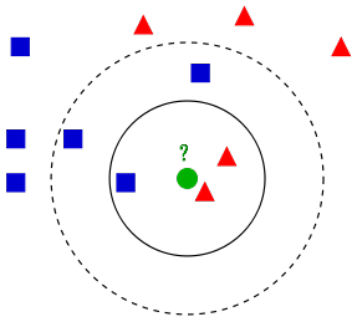
2.2.4 k-Nearest Neighbors (kNN)

La idea de kNN es usar un conjunto de datos etiquetados para predecir la clase de un nuevo elemento en base a los k elementos más cercanos en el espacio de características.

Considere el espacio de características mostrado en la figura 24, se tiene $k = 3$, la

clase 1 (cuadrados) y la clase 2 (triángulos). Se desea estimar la clase del círculo verde. Los $k = 3$ elementos más cercanos a él son los que encuentran en la circunferencia negra.

Figura 24: *Ejemplo de kNN para $K = 3$*

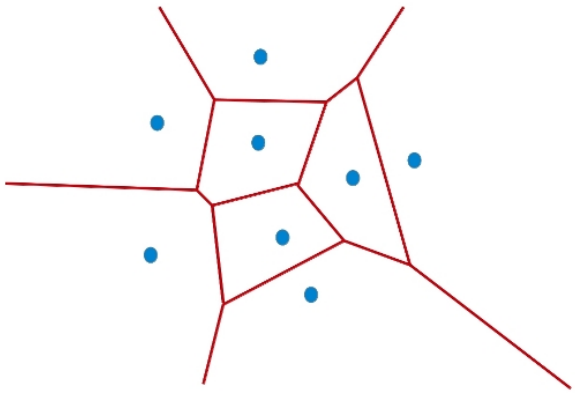


Fuente: *https://goo.gl/7A4ar6*

El círculo verde pertenece a la clase que tenga más elementos cercanos dentro de sus k vecinos. Como existen 2 elementos de la clase 2 y 1 de la clase 1, el círculo verde es de la clase 2. [Leung, 2004]

En el caso particular de $k = 1$, se genera un diagrama de Voronoi (figura 25).

Figura 25: *Diagrama de Voronoi*



Fuente: *https://goo.gl/ZOmJa1*

2.3 Detección de Rostros

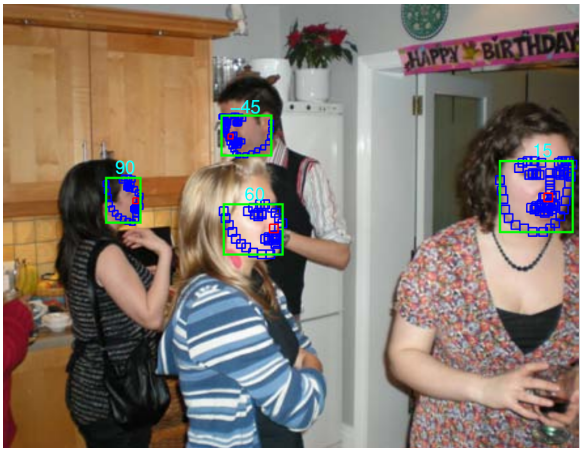
La detección de rostros tiene como objetivo indicar la posición de los rostros y ha sido una de las tecnologías fundamentales para permitir la interacción humano-computador. Mientras que el problema de detección frontal de rostros es ya considerado un problema resuelto, gracias al trabajo inicial de [Viola and Jones, 2001]. La detección con cambio de vista (*multiview face detection*) continúa siendo una tarea desafiante debido a los cambios de apariencia en diferentes poses, iluminación y condición de expresiones.

Las estrategias más clásicas para *multiview face detection* es la de divide y vencerás. En este tipo de enfoques se divide las imágenes en diferentes categorías: frontal, medio perfil, perfil, etc. Se entrenan diferentes clasificadores para cada una de estas categorías. La ventaja de estos métodos es que son rápidos, en particular con cuando se adopta una jerarquía particular. Por otro lado su desempeño no es de lo mejor que existe en la literatura [Hinton, 2002].

Otros métodos aplicados en *multiview face detection* son aquellos que buscan puntos de referencia del rostro, entre ellos el de [Zhu and Ramanan, 2012] que busca estimar la posición basado en el uso de *mixture of trees*. En la figura 26 se observa resultados de este método.

En los últimos años, los métodos de *Deep Learning* han logrado un gran éxito en este problema, logrando detectar casos con oclusiones y en diferentes posiciones, ésto de debe a la disposición de *datasets* lo suficientemente grandes para entrenar estos métodos. En el presente trabajo no se usa estos métodos debido a la limitante de no contar con datos suficientes para entrenar un modelo de manera correcta.

Figura 26: *Resultados de detección de rostros usando Mixture of Trees.*



Fuente: *[Zhu and Ramanan, 2012]*

A continuación se muestra el concepto de *Principal Component Analysis* que será usado en las secciones 2.3.2 y 2.3.3, posteriormente se explica tres métodos que fueron usados como métodos secundarios para la detección de máscaras. El método principal es mostrado en el capítulo 2.4

2.3.1 Principal Component Analysis (PCA)

PCA es un procedimiento estadístico que se basa en la idea de que no toda la información es relevante para un problema específico. Por ejemplo si se tiene una imagen de un rostro de tamaño $p \times q$, no todos los píxeles son igualmente importantes. PCA busca extraer aquellos datos que son más variantes, ya que asume que éstos son los más relevantes. Ésto es bastante útil ya que permite reducir la dimensionalidad de los datos [Bradski, 2000].

PCA es matemáticamente definido como la transformación lineal ortogonal que convierte los datos a un nuevo sistema de coordenadas en el cual, la mayor varianza de alguna proyección de los datos es la primera coordenada del nuevo sistema, la segunda mayor varianza es la segunda coordenada y así sucesivamente.

Asuma que se tiene la matriz de datos x, donde cada fila representa un dato y cada columna es un tipo de característica. Se define la transformación como un conjunto de vectores $p - dimensionales$ $w_{(k)} = (\ w_1, \ldots, w_p)$ que convierte cada vector x_i al vector de componentes principales $t_{(i)} = (\ t_1, \ldots, t_k)$ dado por:

$$t_{k_{(i)}} = x_{(i)}.w_{(k)} \tag{52}$$

2.3.2 Eigenfaces

Este método junto al de la sección 2.3.3 fueron inicialmente planteados para el reconocimiento de rostros. A diferencia de la detección de rostros, el reconocimiento busca identificar a quién pertenece el rostro, mientras que la detección indica la ubicación del rostro. En el presente trabajo se usaron estos métodos para la detección de rostros.

Eigenfaces fue planteado por [Pentland et al., 1994] y calcula la variación en una colección de imágenes para luego utilizar esta información para codificar y compararlas individualmente en una manera holística.

Los *eigenfaces* son el nombre que se le pone a los *eigenvectors* usados como descriptores en *face recognition*. Un *eigenvectors* de una transformación lineal es un vector no nulo que no cambia su dirección cuando se le aplica dicha transformación lineal. Formalmente, si T es una transformación lineal de un espacio vectorial V sobre un campo F de si mismo y v es un vector de V no nulo, v es un *eigenvectors* de T si $T(v)$ es un escalar múltiplo de v. Esta condición es:

$$T(v) = \lambda v \tag{53}$$

Donde λ es un escalar en el campo de F conocido como *eigenvalue*.

Antes de generar los *eigenfaces*, se normaliza para alinear los ojos y bocas, luego se pone todos los datos en la misma resolución. Se extrae los *eigenfaces* de las imágenes mediante el promedio de PCA de la manera siguiente:

- Dado M imágenes de rostros de tamaño $h \times w$, cada imágenes transformada en un vector de tamaño $D = hw$ y se le coloca en un conjunto:

  $$\{\Gamma_1, \Gamma_3, \ldots, \Gamma_M\}$$

  Las imágenes deben estar alineadas y el *background* (partes como el cabello, cuello o fondo de la imagen) debe ser removido.

- Cada rostro difiere del promedio por un vector $\Phi = \Gamma_i - \Psi$, donde el rostro promedio está definido por $\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$.

- La matriz de covarianza $C \in \mathbb{R}^{D\times D}$ se define como:

  $$C = \frac{1}{M}\sum_{i=1}^{M}\Phi_i\Phi_i^T = AA^T \tag{54}$$

  Donde $A = \{\Phi_1, \Phi_2, ..., \Phi_M\} \in \mathbb{R}^{D\times M}$

- Determinar el *eigenvector* de C es una tarea intratable para imágenes de tamaño típico donde $D \gg M$. Sin embargo, para calcular el *eigenvector* de C se puede

calcular primero el *eigenvector* $A^T A$ de una matriz mucho más pequeña $M \times M$. El *eigenvector* y *eigenvalue* $A^T A$ se define como:

$$V = \{v_1, v_3, \ldots, v_r\}$$

Y $\Lambda = diag\{\lambda_1, \lambda_2, \ldots, \lambda_r\}$, $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_r$, donde r es el rango de A. Nótese que los *eigenvector* que corresponden a *eigenvalue* de 0 se descartan.

- Las matrices *eigenvalues* y *eigenvectors* de C son Λ y $U = A\Lambda^{1/2}$, donde $U = \{u_i\}$ es la colección de *eigenfaces*.

[Zhang and Turk, 2008]

2.3.3 Fisherfaces

PCA, el método principal dentro de *eigenfaces*, busca la combinación lineal de características que maximiza la varianza total en los datos. A pesar de que es una forma poderosa de representar los datos, ésta no considera ninguna clase por lo que información importante podría ser perdida. Considérese una situación en la cual la varianza en la data está generada por un factor externo, por ejemplo la luz. Los componentes identificados por PCA no necesariamente contienen alguna información discriminativa por lo que las proyecciones de los datos se unen y la clasificación se hace imposible [Bradski, 2000].

El *Linear Discriminative Analysis (LDA)* realiza una reducción dimensional basado en clases y fue presentado en el trabajo de [Fisher, 1936], los vectores resultantes de LDA son conocidos como *Fisherfaces*.

Para calcular los *Fisherfaces*, se asume que los datos en cada clase son una distribución normal, con promedio μ_i y matriz de covarianza $\sum_i$, luego su función de densidad de probabilidad es $f_i(x|\mu_i, \sum_i)$

Si existen C clases, se tiene $N_i(\mu_i, \sum_i)$ con $i = 1, \ldots, C$. Dada esta distribución normal y las probabilidades previas P_i, la clasificación de un ejemplo x se da comparando las probabilidad *log* de $f_i(x|\mu_i, \sum_i)$ para todo i, esto es:

$$arg\ min_{1 \leq i \leq C} d_i(x) \tag{55}$$

Donde $d_i(x) = (x - \mu_i)^T \sum_i^{-1} (x - \mu_i) + \ln |\sum_i| - 2 \ln P_i$ es conocido como el score discriminativo de cada clase.

Los scores discrimintivos generalmente resultan en límites de clasificación cuadráticos entre clases. Sin embargo, para las clases donde todas las matrices de covarianza son las mismas , $\sum_i = \sum, \forall i$, las partes cuadráticas de d_i se eliminan, produciendo clasificadores lineales. Estos clasificadores reciben el nombre de bases linealmente discriminantes (*linear discriminative bases*), debido a ésto el nombre de LDA. El caso

en el que todas las covarianzas son iguales es conocido como Distribución Normal Homoscedastica.

Si se asume que $C = 2$ y que las clases son Normales Homoscedasticas. Se desea proyectar un vector de características en un espacio uni-dimensional ortogonal al hiperplano de clasificación dados los scores discriminativos. Se nota que el número de ejemplos clasificados de manera incorrecta en el espacio original de p dimensiones y en el subespacio de una sola dimensión es el mismo. Debido a que la clasificación es lineal, todos los ejemplos que estaban de un lado en el plano de p dimensiones, están en el mismo lado en el plano uni-dimensional. Este concepto permite definir LDA y *fisherfaces* [Martinez, 2011].

En general se tiene más de 2 clases, para ésto se formula el problema planteado buscando minimizar la diferencias entre elementos de la misma clase y maximizar entre clases diferentes. Se puede utilizar la matriz de dispersión dentro de la clase para estimar la diferencia:

$$S_w = \sum_{j=1}^{C} \sum_{i=1}^{n_j} (x_{ij} - \mu_j)(x_{ij} - \mu_j)^T \tag{56}$$

Donde x_{ij} es el $i - esimo$ ejemplo de la clase j, μ_j es el promedio de la clase j y n_j es el número de elementos de la clase j.

Igualmente, se puede utilizar la matriz de dispersión entre clases para estimar la diferencias de clases:

$$S_b = \sum_{j=1}^{C} (\mu_j - \mu)(\mu_j - \mu)^T \tag{57}$$

Donde μ representa el promedio de las clases. Ahora se desea encontrar los vectores base V donde S_w es minimizado y S_b es maximizado, donde V es la matriz cuyas columnas v_i son los vectores base que definen el subespacio. Éstas están dadas por:

$$\frac{\left|V^T S_b V\right|}{|V^T S_w V|} \tag{58}$$

La solución para este problema está dada por la descomposición general de *eigenvalues*:

$$S_b V = S_w V \Lambda \tag{59}$$

Donde V es la matriz de *eigenvector* y Λ es la matriz diagonal que corresponde a los *eigenvalues*.

Los *eigenvectors* V asociados a un *eigenvalues* no nulo son los *fisherfaces*. Hay un máximo de $C - 1$ *fisherfaces*. Ésto puede ser deducido de la definición de S_b.

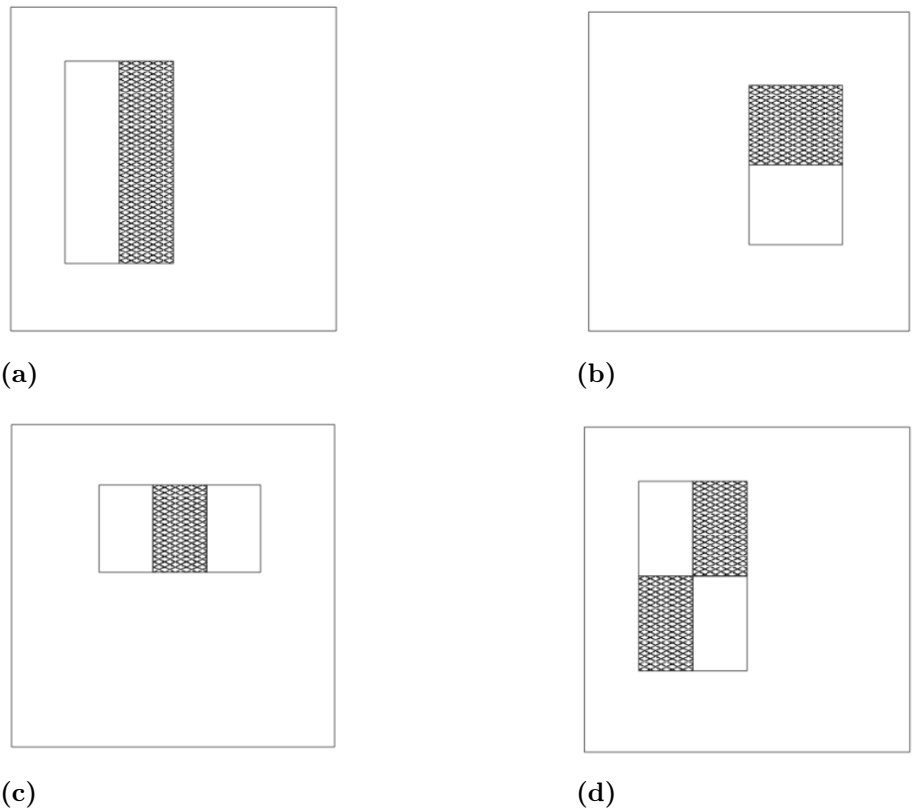
2.3.4 Viola & Jones Cascade

Este método fue introducido por [Viola and Jones, 2001] y está dentro de los más importantes en la detección de rostros debido a la gran cantidad de trabajos que se basaron en esta idea para plantear soluciones para este problema.

El término *cascade* (cascada) se debe a que el método es la combinación de varias etapas, una a continuación de la otra de manera que la salida de una etapa es la entrada de la siguiente. En la primera etapa se calcula la ubicación de los rostros y en las etapas posteriores se van eliminando aquellos que la etapa considere que no contiene rostros.

La detección se hace mediante la comparación de características simples, estas con conocidas como *Haar Features*. El valor de los dos primeros descriptores (figura 27.a y 27.b) es la diferencia entre las sumas de los píxeles dentro de las dos regiones rectangulares. El valor para el tercer tipo de descriptor (figura 27.c) es la suma de los dos rectángulos de fuera menos el rectángulo del centro. Para el cuarto tipo de descriptor (figura 27.d) es la diferencia entre los pares de rectángulos diagonales.

Figura 27: *Haar features*



(a) **(b)**

(c) **(d)**

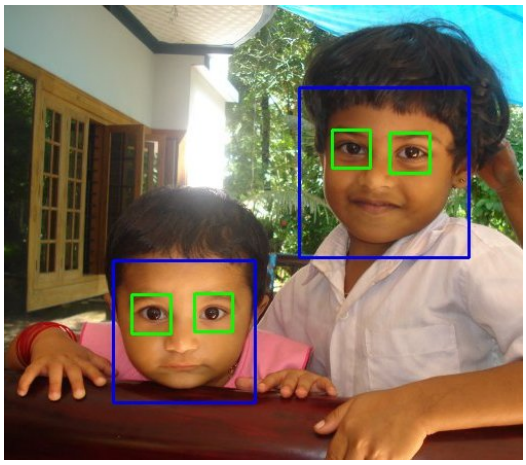
Fuente: *[Viola and Jones, 2001]*

Para calcular los valores de los descriptores de manera rápida [Viola and Jones, 2001] proponen el concepto de *Integral Image* o Imagen Integral. En una imagen integral, cada píxel x, y contiene la suma de todos los píxel desde la esquina superior izquierda hasta x, y, ésto es:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{60}$$

Después de calcular los descriptores se realiza el entrenamiento, para ésto los autores hacen uso del *Adaboost*, esta técnica permite tener un *recall* bastante alto. Este proceso se repite formando una cascada. En cada etapa el umbral se hace más alto y el número de características usadas para la clasificación se hace mayor, ésto permite que falsos positivos sean eliminados en cada etapa. Se pueden ver los resultados de este método en la figura 28.

Figura 28: *Resultados del método planteado por [Viola and Jones, 2001]*



Fuente: *[Bradski, 2000]*

2.4 Histogram of Oriented Gradient

Histogram of Oriented Gradient (HOG) fue presentado por [Dalal and Triggs, 2005], inicialmente se usó para la detección de personas, pero posteriormente fue aplicado a problemas como detección de animales y vehículos.

El método propuesto hace uso de histogramas de la orientaciones de los bordes, el descriptor SIFT y contexto de las formas, pero calculadas en una cuadrícula densa de celdas uniformemente espaciadas y el uso de superposiciones de normalización local de contraste que mejoran los resultados de detección.

La idea básica es que la forma y apariencia del objeto pueden ser caracterizadas por la distribución local de la intensidad de gradiente o la dirección de los bordes, incluso sin conocer la posición del gradiente y borde correspondiente. En la práctica, ésto es implementado como pequeñas regiones de la imagen que reciben el nombre de *celdas*, para cada celda se acumula un histograma 1D. La combinación de histogramas forma la representación final. Para mejorar la invarianza a la iluminación y sombras se plantea normalizar el contraste mediante los histogramas, ésto se logra acumulando la energía en espacios más grandes, conocidos como bloques. A los histogramas de los bloques normalizados, se les conoce como *Histogram of Oriented Gradient* [Dalal and Triggs, 2005].

Se describe a continuación el proceso detallado para calcular HOG:

- **Normalización Gamma/Color:** Antes de procesar los datos, lo autores proponen normalizar los colores de las imágenes. El uso de la normalización gamma de raíz cuadrada mejora sus resultados en un 1%.

- **Cálculo del Gradiente:** La forma en la cual se calcula el gradiente tiene un efecto importante en desempeño del método. Para ésto se usó un filtro Gaussiano con parámetro de suavizado $\sigma = 0$ seguido de un máscara $[-1, 0, 1]$, el uso de otros parámetros demostró empeorar los resultados.

- **Binning de orientaciones:** En esta etapa cada pixel es un voto para un canal del histograma de orientaciones de los bordes. Recordar que los votos son dentro de cada celda. El voto puede ser la magnitud del gradiente, el cuadrado del gradiente u otra función, pero la magnitud directa del gradiente da mejores resultados. Las orientaciones de los canales de los histogramas puede ser entre 0^o y 180^o (gradiente sin signo) o 0^o y 360^o (gradiente con signo), el gradiente sin signo demostró mejores resultados.

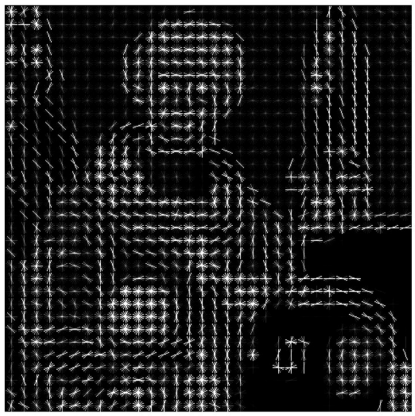
- **Normalización:** Para mejorar los resultados es necesario normalizar dentro de los bloques (conjunto de celdas). Sea v un vector no normalizado, $\|v\|_k$ su

$k-norma$ para $k = 1, 2$ y ϵ un constante pequeña. La forma de normalizar que tiene mejores resultados es:

$$L2 - norm : v \rightarrow v/\sqrt{\|v\|_2^2 + \epsilon^2} \tag{61}$$

En la figura 29 se muestra el resultado de aplicar HOG.

Figura 29: *De izquierda a derecha. Imagen original, imagen después de aplicar HOG*

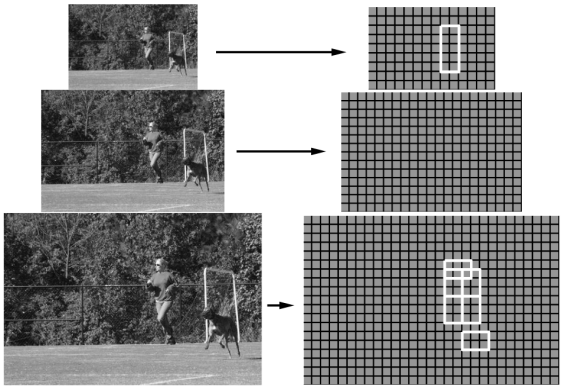


Fuente: *[Van der Walt et al., 2014]*

2.4.1 HOG Pyramid

HOG Pyramid fue planteado para el reconocimiento de personas como parte del método conocido como *Deformed Parts Model* [Felzenszwalb et al., 2010]. La idea es bastante simple, se redimensiona la imagen de entrada a escalas más pequeñas (figura 30) y se computa *HOG* para cada escala. Al final del proceso se unen los resultados de cada escala.

Figura 30: *Ejemplo de HOG Pyramid.*



Fuente: *[Felzenszwalb et al., 2010]*

2.5 Bag of Words (BoW)

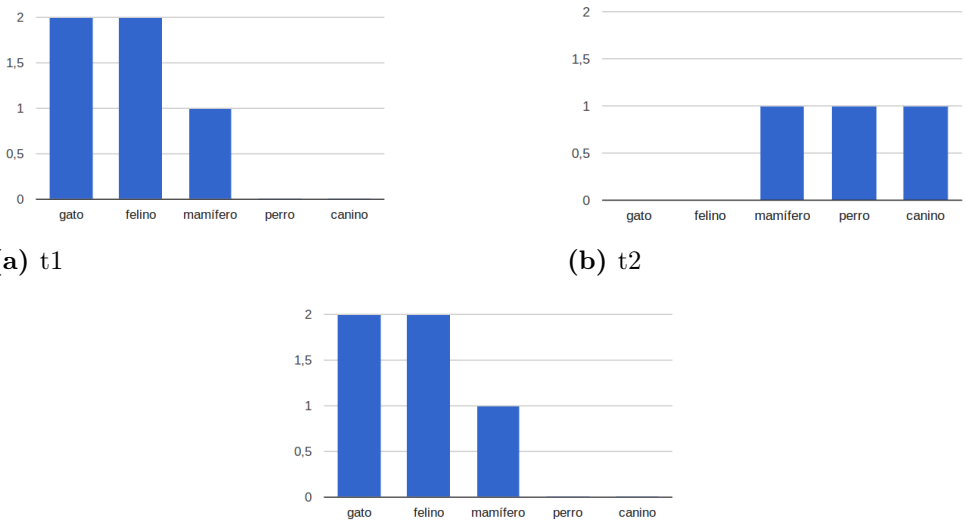
Bag of Words (BoW) es un método bastante popular en la clasificación de imágenes, la idea en la que se centra es bastante fácil de entender en un ejemplo de comparación de textos.

Supóngase que se desea comparar los textos $t1, t2, t3$ de la figura 31.

Figura 31: *Ejemplos de textos*

Una forma simple es contar cuantas veces aparece cada palabra en el texto. A partir de este conteo se puede construir un histograma para cada texto (figura 32), entonces sólo se tiene que comparar estos histogramas para saber si dos textos son iguales. Está claro que esta forma de comparar no considera el orden de las palabras, ya que según este método los textos $t1$ y $t3$ serían iguales, pero no lo son.

Figura 32: *Histogramas de los textos t1, t2 y t3*



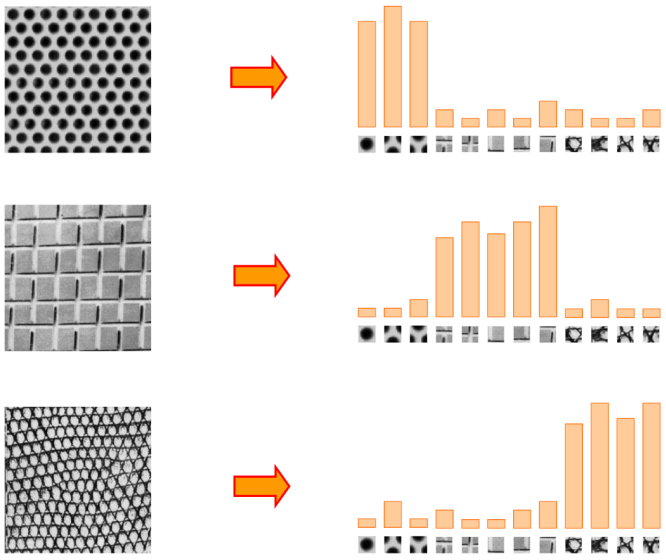
(a) t1

(b) t2

(c) t3

Esta misma idea es usada por BoW para comparar imágenes. Una forma básica de extraer las palabras (*words*) de una imagen es dividirla en pequeñas partes y formar los histogramas a partir de éstas (figura 33).

Figura 33: *Ejemplos de construcción de histogramas para imágenes.*



Fuente: *https://goo.gl/qN7BDg*

A continuación, se explica el proceso de BoW a detalle:

1. **Extracción de Características:** Inicialmente se deben extraer vectores de características de la imagen, una forma que ha mostrado buenos resultados es el uso de descriptores como SIFT y SURF, éstos buscan puntos de interés y calculan vectores de características alrededor de éstos.

2. **Cuantificación:** En esta etapa se convierten los vectores de características en los histogramas equivalentes. El primer paso para lograr ésto es construir el *codebook* que definirá a que *codeword* pertenece cada vector de características. El método más usado para definir el *codebook* es el de *k-Means*, el problema de usar *k-Means* es su complejidad:

$$O(inkd) \tag{62}$$

   Donde i, k, d y n son el número de iteraciones, número de *codewords*, la dimensión de los vectores de características y el número de vectores de características, respectivamente.

   Una forma de solucionar este problema es escoger los *codewords* de manera aleatoria, ésto funciona en la práctica ya que se puede considerar el conjunto de todos los vectores como una distribución normal.

   Una vez que se tiene definido el *codebook* se puede determinar a qué *codeword* están asociados los vectores de características, este proceso es conocido como *coding*. Se procede a unir los resultados del *coding* en los histogramas, este proceso es conocido como *pooling* [Avila et al., 2013]. La forma en la que el *pooling* es realizada tiene efectos importantes en los resultados finales.

   El *sum pooling* es el más clásico y consiste en contar cuántas veces aparece un *codeword* para cada canal del histograma. *Mean pooling* además de contar los *codewords* divide cada canal entre el número de vectores de características.

3. **Clasificación:** Ahora se debe usar los histogramas para entrenar un modelo de clasificación, como se muestra en el trabajo de [Hentschel and Sack, 2014], los SVM con kernel x^2 tiene el mejor desempeño cuando se usa BoW.

Capítulo 3

Desarrollo del Proyecto

3.1 Cusco Typical Dances Dataset

Cusco Typical Dances Dataset es el nombre que se le da al *dataset* creado como parte de este trabajo. Contiene un total de 2560 imágenes que corresponden a las danzas de: Qhapaq Chuncho, Qhapaq Qolla, Negrillos y Contradanza.

Todas estas imágenes corresponden a las Fiestas del Cusco, Corpus Christi y Virgen del Carmen. Existen muchas variantes en el color y forma de los trajes debido al origen de las comparsas, éstos son:

- **Qhapaq Chuncho:** I.E. Inca Garcilazo de la Vega(Provincia de Cusco), Comparsa Qhapaq Chuncho de Huarocondo(Provincia de Anta), Comparsa Qhapaq Chuncho de Paucartambo(Provincia de Paucartambo).
- **Qhapaq Qolla:** I.E. Inca Garcilazo de la Vega(Provincia de Cusco), Comparsa Qhapaq Qolla de Huarocondo(Provincia de Anta), Comparsa Qhapaq Qolla de Paucartambo(Provincia de Paucartambo) y Comparsa Qhapaq Qolla de San Jerónimo(Provincia de Cusco).
- **Negrillos:** Comparsa Negrillo de San Jerónimo(Provincia de Cusco) y Comparsa Negrillo de Paucartambo(Provincia de Paucartambo).
- **Contradanza:** Comparsa Contradanza de San Jerónimo(Provincia de Cusco), Comparsa Contradanza de Paucartambo(Provincia de Paucartambo) y Comparsa Contradanza de Huarocondo(Provincia de Anta).

Se tiene 640 imágenes por danza y cada imagen contiene una única danza[1]. Se muestran entre 1 a 3 danzantes por imagen, éstos están en su mayoría (94%) a cuerpo completo, mientra que el resto muestra entre el 50% y 75% del cuerpo. La característica principal es que las máscaras de los danzantes es visible. Ésto debido a que el enfoque planteado busca estimar la posición del danzante en base a las máscaras en la etapa inicial y ésto es básico para poder clasificar de manera correcta. En la figura 34 se muestran algunos ejemplos de imágenes del *Typical Dance Dataset*.

[1]La clasificación de imágenes es un problema muy complicado. El presente trabajo es uno de los primeros que busca solucionar la clasificación de imágenes de danzas, debido a ésto, no se desarrolla la clasificación multi-etiquetada por la cantidad de imágenes con estas características que serían necesarias recolectar, de igual manera, no se considera una clase neutra ya que eso implica otro problema: dada una imagen, ¿Contiene ésta un danzante?, resolver éste no es objetivo del presente trabajo. Se deja estos problemas para trabajos futuros.

Figura 34: *Imágenes de Typical Dance Dataset*

3.2 Fases del Desarrollo del Proyecto

La clasificación de imágenes implica predecir la clase (nombre) de la danza que se presenta en una imagen de consulta. Además, la literatura muestra que la detección, como etapa previa para la clasificación, es fundamental para sistemas que buscan solucionar este problema [Parekh et al., 2014].

3.2.1 Descripción de las fases

El problema será abordado en 2 fases (figura 35):

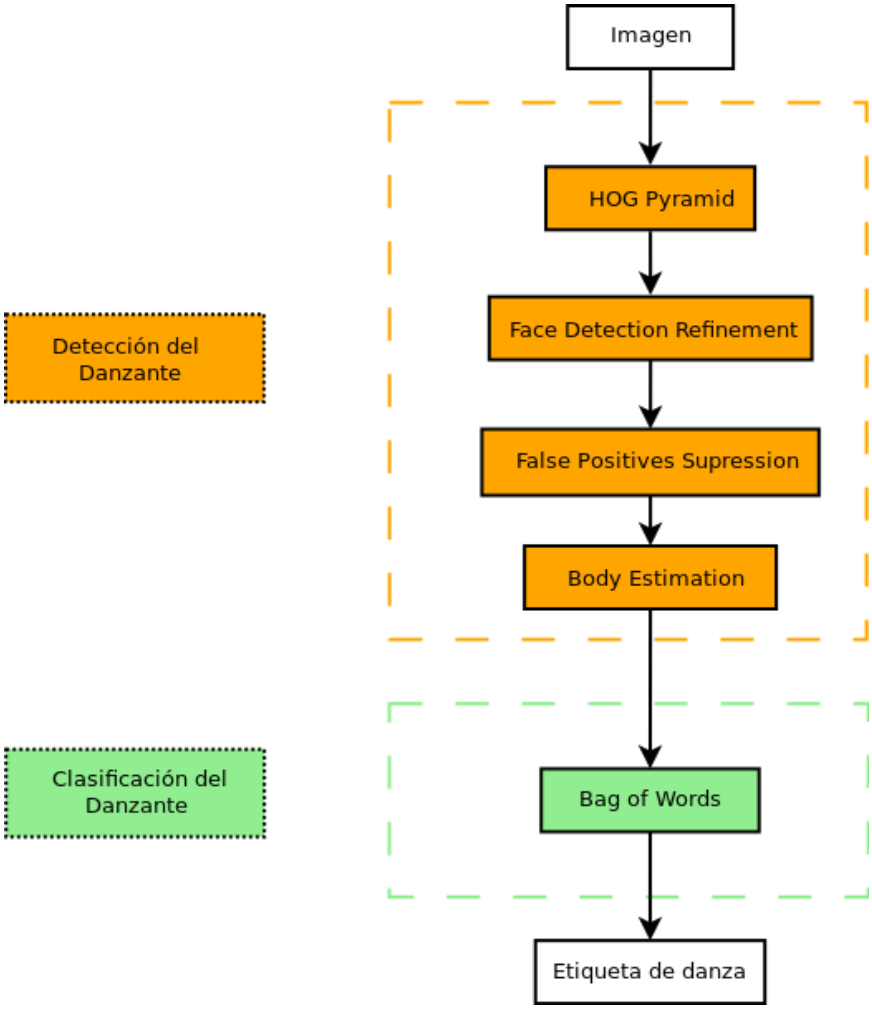
- La primera fase (Detección de Danzantes) indica en qué partes de la imagen se encuentran los danzantes.
- La segunda fase (Clasificación de Danzantes) predice a qué clase pertenece la imagen.

Si bien estas dos fases son generales en cualquier sistema que busque clasificar imágenes, los pasos dentro de cada una de ellas varía dependiendo del problema. En el presente trabajo, el método usado para la detección del danzante es de autoría propia, mientras que en la clasificación se usa un método ampliamente usado en la literatura.

La primera fase busca eliminar toda aquella información que no es relevante para la clasificación. Debido a que hay casos en los cuales existen personas que no bailan dentro de la imágenes, se busca estimar la posición del danzante en base a la ubicación de las máscaras; se usa esta idea ya que si se considera todo el cuerpo, diferenciar entre danzante y no danzante se torna más complicado. Ésto debido a que el espacio de variaciones del cuerpo completo es mayor al de un rostro.

Una vez estimada las posiciones iniciales mediante *HOG Pyramid*, se procede a refinar los candidatos detectados usando *Hough Transform*. Este proceso permite que los rostros que hayan sido detectados parcialmente puedan tener una mejor estimación. Continuando con esta fase, se eliminan falsos positivos mediante técnicas más clásicas de detección de rostros. Finalmente se estima la posiciones de los danzantes en base a los candidatos que hayan superado la eliminación de falsos positivos.

Figura 35: *Fases del proyecto.*



La segunda fase usa en los resultados obtenidos en la fase previa para entrenar el método de *Bag-of-Words (BOW)* y así construir un modelo para la clasificación de imágenes. Para ésto se separa los datos en un conjunto de entrenamiento y otro de evaluación. Como se explica en el marco teórico, se construye el *codebook* de manera aleatoria y se usan descriptores *SIFT* y *SURF* para extraer la información relevante de las imágenes.

Este enfoque genera una limitante debido a la forma en la cual se detecta los danzantes. Como se dijo en la sección 1.1.6, la clasificación dentro de la tasa de acierto mostrada en los resultados se limita a imágenes que cumplen con las características del *dataset* que será mostrado en el capítulo 3.1, ésto no impide que imágenes que no cumplen estas características puedan ser clasificadas de manera satisfactoria por el método propuesto.

3.2.2 Detección de Danzantes

Esta etapa es importante debido a que se elimina información que no es relevante para el proceso de clasificación. La entrada para este proceso son las imágenes de las danzas y la salida son las regiones de las imágenes que contienen a los danzantes.

3.2.2.1 HOG Pyramid

HOG Pyramid es usado en el presente trabajo para evitar que cambios de escalamiento tengan efectos en la detección del danzante.

Para detectar los danzantes se ejecuta un *slide windows* de 50×50 sobre la imagen y mediante el uso del **Clasificador** ϖ se determina la etiqueta de cada ventana (0 si no contiene máscara, 1 si contiene parte de una máscara), este proceso se realiza para cada escala:

```
1: function DETECTAR MÁSCARAS (imagen)
2:     for Escala en HOG Pyramid do
3:         Inicializar matriz DetectedFaces con 0's
4:         for ventana en imagen_Escala do
5:             label = ϖ(HOG(ventana))
6:             DetectedFaces[ventana] = DetectedFaces[ventana] + label
7:         end for
8:     end for
9: end function
```

Fuente: *Elaboración propia*

Donde la función $HOG(ventana)$ calcula HOG para la ventana. El elemento más importante en la función *Detectar Máscaras* es el **Clasificador** ϖ, a continuación se explica a detalle el proceso de entrenamiento de este modelo.

Construcción de datos positivos y negativos

Dado que se desea construir un modelo que clasifique ventanas de 50×50, se debe construir datos positivos (ventanas con máscaras) y negativos (ventanas sin máscaras) del mismo tamaño para ser usados en el entrenamiento. Lo primero que se tiene que hacer es indicar en que partes de la imagen se encuentran las máscaras, este proceso es manual y consiste en construir una imagen binaria I_{mask} cuyos píxeles (i, j) que sean parte de la máscara en la imagen original I tienen valor 1 (negro), caso contrario valor 0 (blanco):

$$I_{mask}(i, j) = I(i, j) \in Máscara?1 : 0 \tag{63}$$

En la figura 36 se observa el resultado de este proceso.

Figura 36: *De izquierda a derecha. Imagen Original, imagen binaria de la máscara.*

En el proyecto se usó el 19.06% (488) imágenes para este proceso. En base a estas nuevas imágenes I_{mask} se pueden construir los datos mediante un *slide windows* que recorra cada imagen I y que en base a I_{mask} etiquete cada ventana con un valor entre 0 y 100. Para acelerar este proceso de cálculo y evitar que la complejidad total sea $O(n^2m^2)$, donde n y m son las dimensiones de la imagen I. Se hace uso del concepto de *Integral Image* o Imagen Integral ii planteado por [Viola and Jones, 2001] (sección 2.3.4).

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{60}$$

Ésto permite que la complejidad total de crear los datos positivos y negativos sea $O(nm)$. Este proceso también se realiza para cada escala:

```
1: function CREAR DATOS POSITIVOS Y NEGATIVOS (I, I_mask)
2:     for Escala en HOG Pyramid do
3:         for ventana en I_{mask,Escala} do
4:             label = ξ(ventana)
5:             guardar(I[ventana], label)
6:         end for
7:     end for
8: end function
```

Fuente: *Elaboración propia*

La función $\xi(ventana)$ retorna un valor entre 0 y 100 (mientras más grande este valor, más factible usar la ventana como positivo). Ésto es importante ya que determinará si un ventana es considerada positiva y negativa. $\xi(ventana)$ se basa en el porcentaje de la ventana que es máscara y en el porcentaje del máscara que se encuentra en la ventana, estos porcentajes se calculan en $O(1)$ gracias al uso de *Integral*

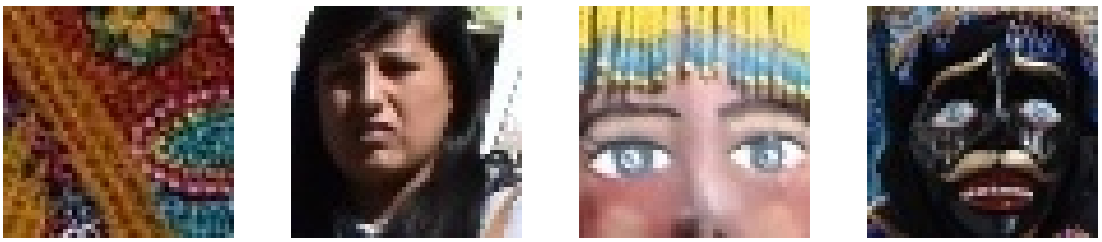
Image:

1: **function** $\xi(ventana)$
2: $windowPercentage = \frac{100 \times |(i,j) \in Máscara, Ventana|}{|(i,j) \in Ventana|}$
3: $facePercentage = \frac{100 \times |(i,j) \in Máscara, Ventana|}{|(i,j) \in Máscara|}$
4: **if** windowPercentage <= 25 and not facePercentage >= 50 **then**
5: *return* $windowPercentage$
6: **end if**
7: *return* $max(windowPercentage, facePercentage)$
8: **end function**

Fuente: *Elaboración propia*

En la figura 37 se observa algunos ejemplos de ventanas y sus valores de $\xi(ventana)$.

Figura 37: *Ejemplos de datos negativos y positivos.*



(a) $\xi = 0$ **(b)** $\xi = 0$ **(c)** $\xi = 98$ **(d)** $\xi = 99$

Entrenamiento del Clasificador

Una vez que se tiene los datos positivos y negativos se procede a entrenar el **Clasificador ϖ. Se realizan pruebas con tres clasificadores ampliamente usados en la literatura: Adaboost, Random Forest y Support Vector Machine (SVM)**[1]. Se define un $umbral = 30$ para poder separar datos positivos de negativos en base al valor de $\xi(ventana)$.

Debido a que existen más datos negativos que positivos, sólo una pequeña porción de la imagen contiene máscaras, y para evitar el problemas de clases no balanceadas [2], se realiza un proceso de selección de datos.

Se limita el número de datos de entrenamiento al tamaño de clase positiva, ya que ésta es siempre menor. Se extrae *HOG* para cada ventana y en base a todos éstos se entrena el *Clasificador* ϖ.

[1]Los efectos de cada uno de éstos se muestran en el capítulo de resultados.

[2]Cuando existe mucha diferencia entre los tamaños de los datos por clase y por la forma de la función objetivo de los modelos, se genera una sobre entrenamiento para alguna clase. `https://goo.gl/Qthq9f`

Los parámetros usados para *HOG*, *Adaboost*, *Random Forest* y *SVM* se muestran en la tabla 5.

Tabla 5: *Parámetros usados para HOG y Adaboost.*

HOG	
Parámetro	**Valor**
tamaño de celdas	2×2
tamaño de bloques	9×9
# de orientaciones	9

ADABOOST	
Parámetro	**Valor**
# de estimadores	150
Tipo de estimadores	Decision Tree

RANDOM FOREST	
Parámetro	**Valor**
# de estimadores	150
criterion	Gini

SUPPORT VECTOR MACHINE	
Parámetro	**Valor**
Kernel	rbf
C	100
γ	0.001

Una vez detectadas máscaras mediante *HOG*, para cada imagen se tienen dos matrices (imágenes *HOG*) que corresponden a la detección de las 2 primeras escalas $(1:1\ ,\ 1:2)$. En la figura 38 se muestran algunos de los resultados, las regiones de color rojo son las que el método considera que presentan máscaras, las regiones de color azul indican lo contrario.

Se puede ver que en las figuras 38 y 39, existen área que se consideran que tienen máscaras mas no las tienen. Todos estos falsos positivos serán eliminados en las etapas siguientes.

Figura 38: *De izquierda a derecha. Imagen original, imagen HOG para la primera escala, imagen HOG para la segunda escala.*

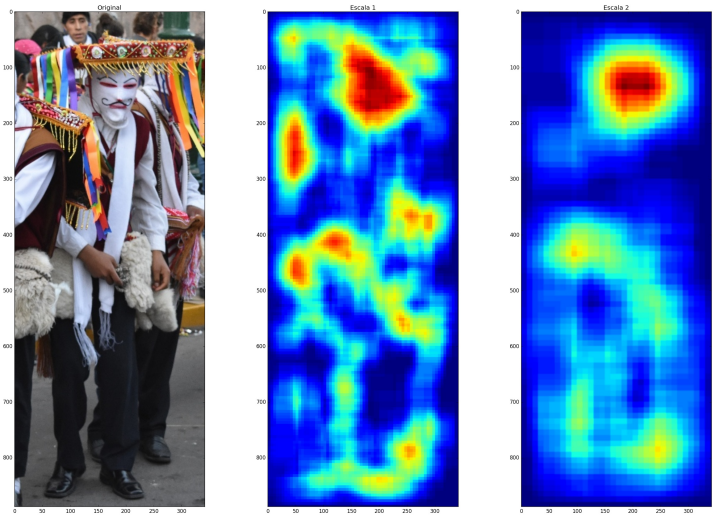
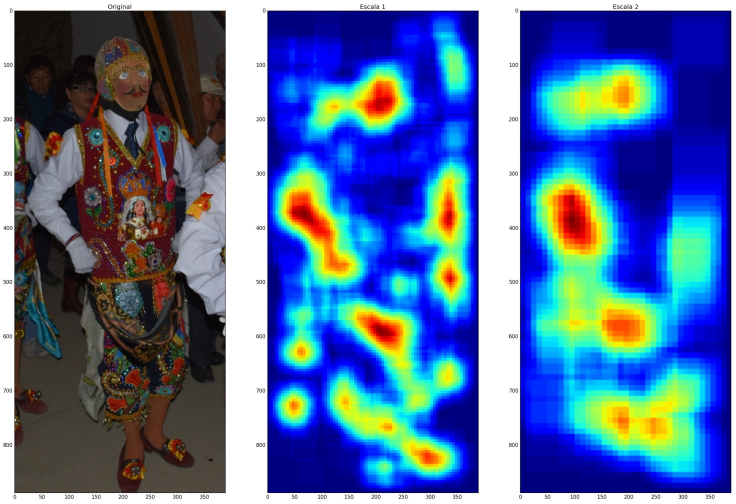


Figura 39: *De izquierda a derecha. Imagen original, imagen HOG para la primera escala, imagen HOG para la segunda escala.*



Para finalizar esta etapa se computa un algoritmo de recorrido en grafo sobre las imágenes de las dos escalas, se añade la regla:

1: **function** ISVALID(x, y)
2: $\quad$ *return* $Matrices_{HOG}(Escala)_{(x,y)} \geq threshold$ *and* $(x, y) \notin Lista_{Visitados}$
3: **end function**

Fuente: *Elaboración propia*

El valor de *threshold* varía para cada escala. Es 70% del valor máximo de la matriz para la primera escala y de 50% para la segunda. El algoritmo es ahora:

```
1  def bfs(i, j, maxI, maxJ, minI, minJ, threshold, visited,
       dfsImg):
2      visited[i][j] = 1
3      q = Queue.Queue(maxsize=0)
4      q.put((i, j))
5      while(not q.empty()):
6          cur = q.get()
7          visited[cur[0]][cur[1]] = 1
8          maxI = max(maxI, cur[0])
9          maxJ = max(maxJ, cur[1])
10         minI = min(minI, cur[0])
11         minJ = min(minJ, cur[1])
12         for k in range(8):
13             ni = cur[0] + dx[k]
14             nj = cur[1] + dy[k]
15             if(isValid(ni, nj)):
16                 visited[ni][nj] = 1
17                 q.put((ni, nj))
18 return [minJ, minI, maxJ, maxI], visited
```

Fuente: *Elaboración propia*

Este algoritmo de búsqueda en anchura genera componentes conexas teniendo como base el *threshold* previamente indicado. El resultado del BFS se muestra en la figura 40. Se puede ver que existen muchos falsos positivos dentro de la detección, ésto se debe a que el *umbral* definido para generar datos positivos y negativos es bastante bajo ($umbral = 30$). El uso de umbrales más altos ocasiona que el proceso de detección elimine también candidatos que contienen máscaras.

Lo que se busca con un umbral bajo es tener un *recall* alto (que la mayor cantidad de máscaras sean detectadas), ésto no implica que todos los resultados tengan que ser máscaras.

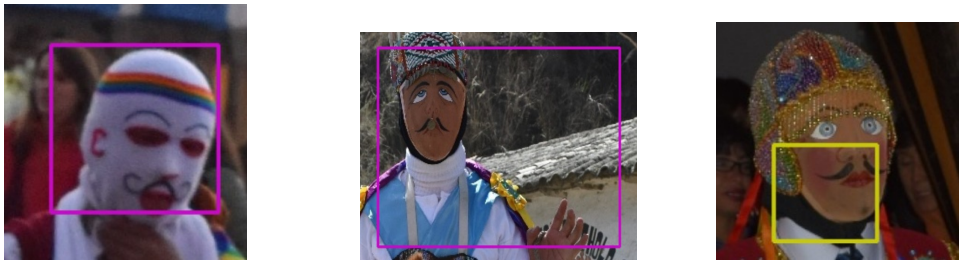
Figura 40: *De izquierda a derecha. Resultados para la primera escala, resultados para la segunda escala.*

3.2.2.2 Face Detection Refinement

Los resultados de la sección anterior se pueden dividir en dos tipos: candidatos que contienen parte o toda la máscara (candidatos positivos) y las que no contienen nada de las máscaras (candidatos negativos).

Los candidatos positivos se pueden dividir a su vez en: candidatos que contienen toda la máscara y los que contienen sólo una parte. Los candidatos que contienen toda la máscara pueden contener también más de lo que se desea (figura 41).

Figura 41: *De izquierda a derecha. Detección correcta, detección con exceso y detección parcial.*



(a) **(b)** **(c)**

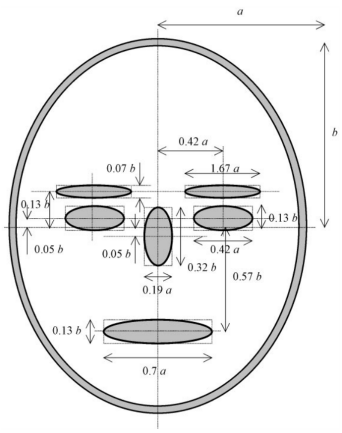
Para poder eliminar los candidatos negativos sin eliminar los candidatos positivos, es necesario que la detección sobre las máscaras no presente los casos de la figura 41.b y 41.c.

El objetivo de esta fase mejorar las detecciones de *HOG pyramid* para evitar que los candidatos positivos sean eliminados en la siguientes fases.

En la detección de rostros existen un conjunto de enfoques de tipo geométrico, en éstos se busca explotar características como las formas y las distancias que existen entre las ojos, boca y nariz. Dentro de este tipo de enfoques está el trabajo de [Maio and Maltoni, 2000], los autores proponen que es posible detectar rostros usando *Hough Transform* para detectar elipses, ya que un rostro tiene una forma aproximada a esta forma geométrica (figura 42).

Se usa esta idea pero buscando círculos mediante *Hough Transform*. Se establece también que los círculos pueden tener el centro fuera del área de detección, ésto permitirá que casos que tengan una detección parcial sean mejorados.

Figura 42: *Modelo geométrico de rostro.*



Fuente: *[Maio and Maltoni, 2000]*

Detección de bordes

Antes de usar *Hough Transform* para la detección, es necesario calcular los bordes. El resultado de aplicar *Canny Edge Detector* de manera directa se muestran en la figura 43.

Figura 43: *Detección de bordes sin preprocesamiento.*

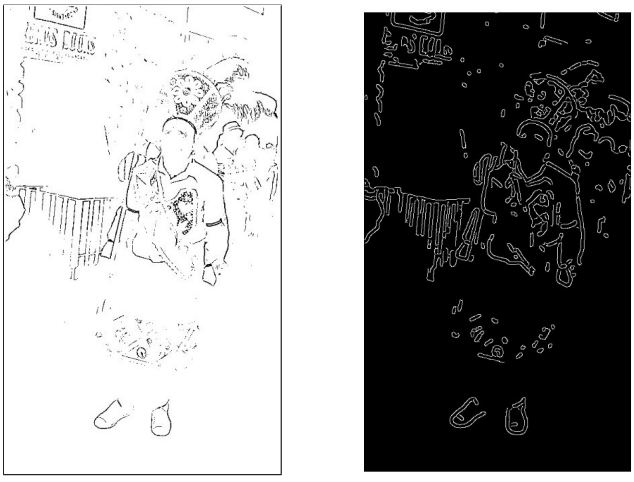
Se puede observar en la figura 43 que existe mucho ruido, ésto hará que los resultados de la detección de círculos no sea de manera correcta sobre la máscara. Para eliminar el ruido se realiza un preprocesamiento[3] a la imagen (figura 44):

[3]Al igual que el método presentado por [Abo-Zahhad et al., 2014], se busca suprimir partes irrelevantes de la imagen. Los pasos del preprocesamiento son bastante usados en la literatura.

1. Se usa *bilateral filter* para poder suavizar y eliminar el ruido de la imagen, este filtro es bastante usado en la literatura para este fin.
2. Se convierte la imagen del sistema RGB al HSV y se extrae el canal V (escala de blanco a negro), se utiliza HSV porque separa la intensidad de los componentes del color, ésto genera robustez frente a cambios de iluminación y sombras.
3. Se realiza *adaptive thresholding* para binarizar la imagen, este método permite una mejor binarización frente al *simple thresholding*.

Figura 44: *Resultados de cada etapa del preprocesamiento.*

(a) Imagen original **(b)** Bilateral Filter **(c)** Canal V



(d) Thresholding **(e)** Bordes

Una vez terminado el preprocesamiento se usa Canny para detectar los bordes. Se puede ver la diferencia entre la detección de bordes con preprocesamiento y sin preprocesamiento en la figura 45.

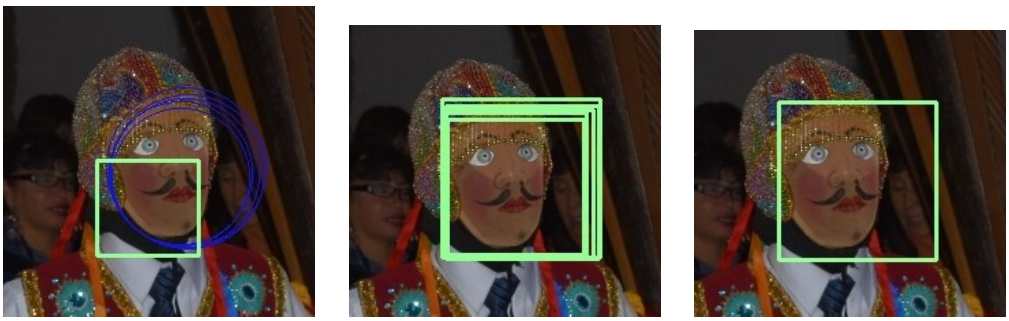
Figura 45: *Comparación de resultados.*

(a) Con preprocesamiento. **(b)** Sin preprocesamiento.

Detección de círculos

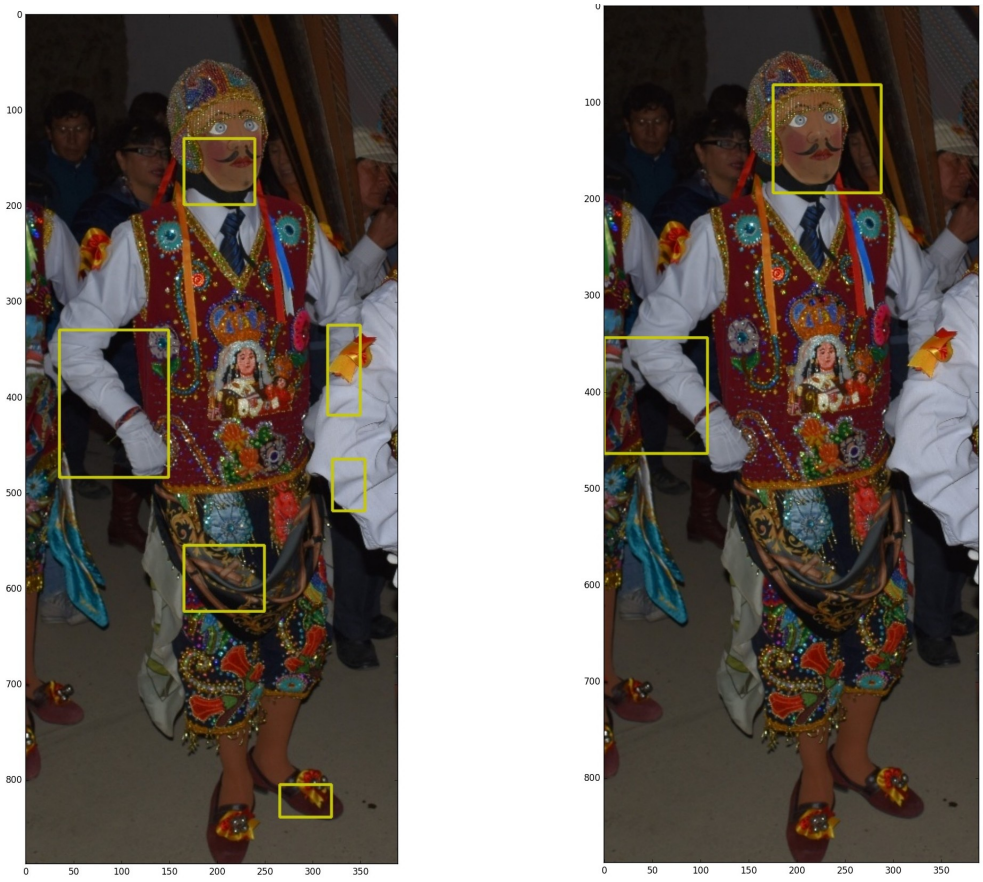
Una vez detectado los bordes, se aplica *Hough Transform* para detectar 5 círculos por candidato. Para cada círculo se calcula el cuadrado circunscrito y como se puede ver en la figura 46 muchos de ellos se solapan por lo que es necesario usar *non-maximun Suppression (nms)* para eliminar el solapamiento.

Figura 46: *De izquierda a derecha. Detección de círculos, cuadrados circunscritos y resultados de nms.*



Los resultados es esta fase se pueden ver en la figura 47.

Figura 47: *Resultados de Face Detection Refinement (FDR)*



(a) Antes de FDR

(b) Después de FDR

3.2.2.3 False Positives Suppression

En esta fase se busca eliminar la mayor cantidad de falsos positivos. Para ésto se usa: *Eigenfaces*, *Fisherfaces* y *Viola & Jones Cascade*.

Eigenfaces y *Fisherfaces* son de usados de manera similar, se realiza un *slide windows* sobre cada candidato y para cada ventana se calcula la etiqueta de clase del método correspondiente (1 para máscaras y 0 en caso contrario). Todos aquellos candidatos que con ninguna ventana hayan obtenido una etiqueta diferente de 0 son eliminados, este paso es también independiente para cada escala.

```
1: function FPS Eigenfaces y Fisherfaces(lista_candidatos)
2:     for Escala en HOG Pyramid do
3:         for Candidato en lista_candidatos do
4:             isDeleted = True
5:             for ventana en imagen_Escala do
6:                 labelEigen = Eigenfaces(ventana)
7:                 labelFisher = Fisherfaces(ventana)
8:                 if labelEigen == 1 or labelFisher == 1 then
9:                     isDeleted = False
10:                end if
11:            end for
12:            if isDeleted == True then
13:                eliminar(Candidato)
14:            end if
15:        end for
16:    end for
17: end function
```

Fuente: *Elaboración propia*

Las funciones $Eigenfaces(ventana)$ y $Fisherfaces(ventana)$ corresponden a los métodos mostrados en las secciones 2.3.2 y 2.3.3. Estos métodos requieren ser entrenados de manera similar a *HOG Pyramid*.

Se hace uso de los mismos datos negativos usados en *HOG Pyramid* pero los datos positivos son ahora imágenes que contienen en su totalidad rostros (figura 48). Ésto se hace con la idea de ser más discriminativos en la detección.

Figura 48: *Ejemplos de datos positivos para False Positives Suppression.*

Viola & Jones Cascade es por sí un método de detección de rostros. Para utilizarlo como filtro, se entrena el modelo planteado por los autores pero haciendo uso de descriptores *LBP* (originalmente se hace uso de descriptores *Haar*[4]). Los datos positivos y negativos son los mismos usados para entrenar *Eigenfaces* y *Fisherfaces*.

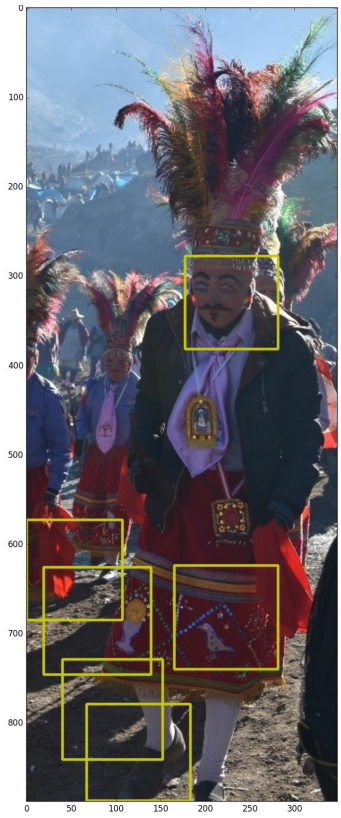
[4]Se usa este descriptor debido al alto costo computacional que implica el entrenamiento con Haar features.

Una vez que se tiene entrenado el modelo, se detectan las máscaras. Todos aquellos candidatos (en las 2 escalas) que no se intersectan con alguna detección del método *Viola & Jones Cascade* (figura 49) son eliminados.

Figura 49: *Resultados de Viola & Jones Cascade*

Para finalizar esta fase, se aprovecha las características de las imágenes para eliminar aquellos candidatos que se encuentren muy pegados a los bordes izquierdo, derecho e inferior. Los resultados de esta fase se muestran en la figura 50.

Figura 50: *Resultados de False Positives Suppression (FPS).*



(a) Antes de FPS

(b) Después de FPS

(c) Antes de FPS

(d) Después de FPS

3.2.2.4 Body Estimation

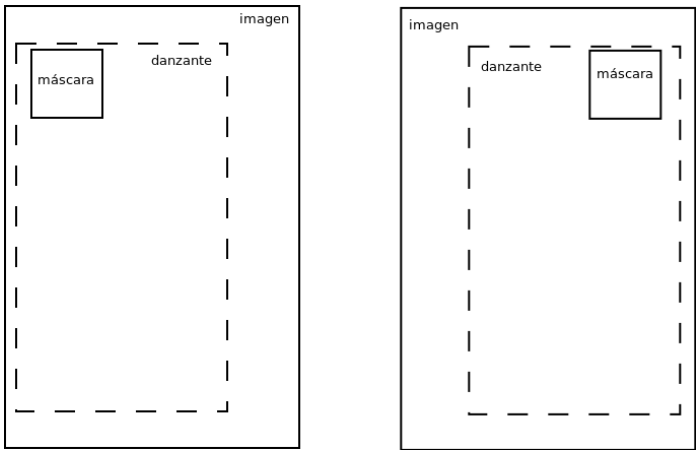
Una vez que se tiene los candidatos de máscara se debe estimar al danzante bajo la siguiente premisa: El cuerpo de un danzante siempre se encuentra en una posición inferior a la máscara. La forma de estimación sigue la heurística que se muestra a continuación.

Dado que queremos encontrar una región que cubra al danzante se deben definir los límites de la región. El límite superior es el más simple de determinar ya que está limitado por la detección de la máscara.

Los límites izquierdo y derecho se determinan en base a la forma de las imágenes: se asume que $ancho_{danzante} = 0.75 \times ancho_{imagen}$. En un caso general es posible fijar $ancho_{danzante}$ a un valor constante, pero si se observa la forma de los datos queda claro que considerar el factor 0.75 permite estimar de buena manera el valor de $ancho_{danzante}$.

Si se ve el cálculo de los límites izquierdo y derecho como un proceso de extensión del área de la máscara detectada; se puede ver que si la detección se ha realizado más hacia la izquierda, entonces se debe extender el ancho en mayor porcentaje hacia la derecha. De igual manera, si la detección se hizo más hacia la derecha entonces la extensión se hace en mayor porcentaje hacia la izquierda (figura 51).

Figura 51: *Idea de estimación de danzante.*



Se pueden diferenciar 3 casos: cuando la detección de la máscara está a la izquierda del medio de la imagen, cuando está a la derecha y cuando está sobre el medio. Sean m y n los límites izquierdo y derecho de la detección de máscara, se tiene el siguiente algoritmo que calcula los límites izquierdo y derecho del danzante:

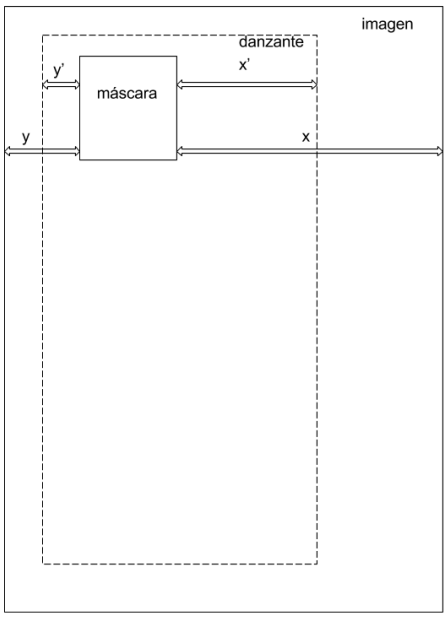
function ESTIMAR LÍMITES IZQUIERDO Y DERECHO(n, m)
    $ancho_{restante} = 0.75 \times ancho_{imagen} - ancho_{detección}$
    if $m \leq medio$ **then**
        $x = m$, $y = medio - n$
    else if $n \geq medio$ **then**
        $x = m - medio$, $y = ancho_{imagen} - n$
    else
        $x = m$, $y = ancho_{imagen} - n$
    end if
    $x' = \frac{ancho_{restante}}{1+x/y}$
    $y' = ancho_{restante} - x'$
    $\text{límite}_{izquierda} = n - x'$
    $\text{límite}_{derecha} = m + y'$
end function

Fuente: *Elaboración propia*

Estas fórmulas salen de la relación 64 mostrada en la figura 52.

$$\frac{x}{x'} = \frac{y}{y'} \tag{64}$$

Figura 52: *Imagen de relaciones entre distancias.*



Para estimar el límite inferior se hace uso de concepto de entropía. Si se consid-

era que las danzas se realizan sobre diferentes tipos de suelo, éstos al ser un tanto uniformes presentan valores bajos de entropía. En la figura 53 se puede ver el cálculo de entropía sobre una imagen.

Figura 53: *Cálculo de entropía y imagen con límite inferior calculado (línea verde).*

Los valores de la entropía varían entre 0 y 7 aproximadamente, lo que se hace es establecer una línea horizontal y calcular:

$$low_{entropy} = \sum entropy/entropy \leq 1 \tag{65}$$

$$high_{entropy} = \sum entropy/entropy \geq 3 \tag{66}$$

Desde la línea horizontal l hasta el final de la parte baja de la imagen.

Mientras $low_{entropy} \geq 40\%$ *de* $entropy_{total}$ y $high_{entropy} < 25\%$ *de* $entropy_{total}$ se mueve la línea horizontal hacia arriba. Al finalizar este proceso, el límite está definido por:

$$\frac{largo_{imagen} - l}{2} \tag{67}$$

Ahora ya se tiene definido la forma en la cual se estima el cuerpo.

Se puede ver que en los resultados de la fase anterior aún existen falsos positivos. Mediante la estimación de cuerpo se eliminará finalmente todos éstos.

Antes de iniciar el proceso de estimación final se deben juntar los resultados de ambas escalas (figura 54).

Figura 54: *Resultados de unión de escalas, escala* $1:1$ *en amarillo y* $1:2$ *en magenta.*

Una vez que se tienen los resultados de ambas escalas combinados (*máscaras*) se computa el siguiente algoritmo:

function UNIR RESULTADOS(*máscaras*)
    $area_{max} = 0$
    $resultado = null$
    while *máscaras* $\neq \emptyset$ **do**
        $ordenar(máscaras)$
        $current = máscaras[0]$
        $dancer = estimar_{cuerpo}(current)$
        $máscaras.push(dancer)$
        $Non - Maximum\ Suppression(máscaras)$
        if $area(dancer) > area_{max}$ **then**
            $resultado = dancer$
            $area_{max} = area(dancer)_{max(limit_{upper})}$
        end if
        $máscaras.delete(dancer)$
    end while
    return resultado
end function

Fuente: *Elaboración propia*

Lo que hace el algoritmo es repetir mientras existan máscaras:

- Ordenar las máscaras de acuerdo a sus componentes (i, j) (de arriba hacia abajo, de izquierda a derecha).
- Escoger la primera máscara en la variable *current* y estimar el cuerpo en base los pasos mostrados anteriormente.
- Insertar el cuerpo estimado.
- Realizar el algoritmo de $Non - Maximum\ Suppression$ para eliminar todos aquellos candidatos de máscara que se encuentran dentro de cuerpo estimado. Ésto hace que el número de máscaras disminuya y llegue a ser 0 en algún momento.
- Actualizar el valor de *resultado* con el danzante cuya estimación tenga mayor área. Se mantiene como límite superior de *resultado* el máximo limite superior entre todas las detecciones de máscaras.

Con esta fase se termina la detección de danzante. En la figura 55 se observan resultados de esta etapa.

Figura 55: *Resultados de la detección de danzantes.*

3.2.3 Clasificación de Danzantes

La entrada para esta etapa son las regiones de las imágenes resultado de la detección de danzantes. La salida es la etiqueta de la danza predicha.

Para la clasificación se hace uso del método de *Bag of Words (BoW)*. Este método se divide en 3 fases: Extracción de características, cuantificación y clasificación. En las 2 últimas fases es necesario construir modelos no supervisados y supervisados, correspondientemente.

3.2.3.1 Extracción de Características

Para iniciar con el proceso se deben extraer vectores de características (*features*) de las imágenes. En el caso de la danzas, se deben buscar *features* que puedan describir las imágenes de manera que las variaciones de posición, color e iluminación de la danza no generen problemas.

Los descriptores locales han demostrado ser invariantes a rotaciones, escalamiento, iluminación, ruido y cambios de vista poco bruscos. A diferencia de descriptores globales, un descriptor local extrae información sólo de algunas partes de la imagen (puntos de interés), éstas características hacen que este tipo de descriptores sean adecuados para el presente trabajo.

La literatura presenta a *SIFT* [Lowe, 2004] y *SURF* [Bay et al., 2006] como los descriptores más populares. Se usan ambos descriptores, cada uno por separado, y se comparan los resultados que producen.

Se puede definir un punto de interés como una región de la imagen que contiene información relevante, a cada punto de interés existen asociados un conjunto de vectores de características:

$$v_i = \{a_1, \ldots, a_d\}, i \in 1, \ldots, n \tag{68}$$

Donde n es el número de vectores de características para el punto de interés [5] y d es el número de dimensiones para cada vector [6]. En la figura 56 se pueden ver los puntos de interés para *SIFT* y *SURF*

[5] Este valor depende del tamaño del punto de interés.

[6] Este valor es 128 para *SIFT* y 64 para *SURF*.

Figura 56: *Puntos de interés detectados.*

(a) Original **(b)** SIFT **(c)** SURF

3.2.3.2 Cuantificación

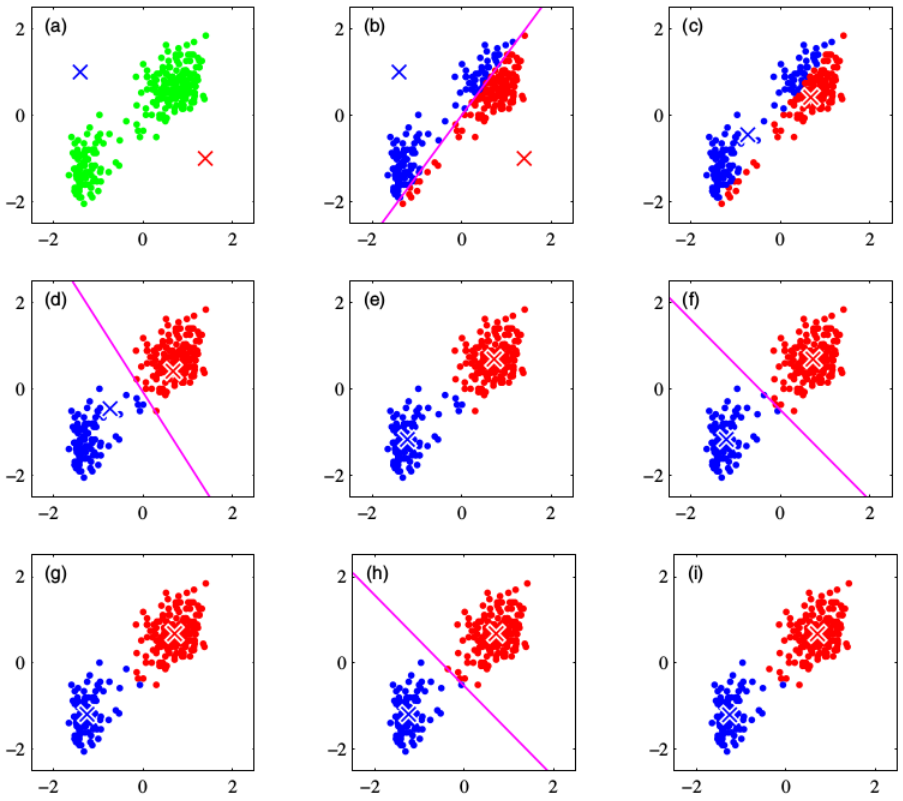
En esta fase se calcula los histogramas asociados a cada punto de interés. Para ésto, se debe construir un *codebook* y en base a este modelo calcular los histogramas.

Construcción de Codebook

El *codebook* es el diccionario visual, está formado por el conjunto de *codewords* (palabras) que definen los histogramas.

Considérese que la dimensión de los vectores de características igual a 2 ($k = 2$), y se desea construir el *codebook*. Si se tiene que los datos son los mostrados en la figura 57.a (puntos verdes), se puede usar el algoritmo de *k-Means* para clusterizar los datos (figura 57.i). Finalmente los centroides son considerados los *codewords*.

Figura 57: *Construcción de codebook mediante k-Means.*



Fuente: *[Bishop, 2006]*

El uso de *k-Means* es bastante popular para la construcción de *codebooks*, pero su principal desventaja es el costo computacional del algoritmo:

$$O(inkd) \tag{62}$$

Donde i, k, d y n son el número de iteraciones, número de *codewords*, la dimensión de los vectores de características y el número de vectores de características, respectivamente.

Dado que los datos pueden ser considerados como una distribución normal (no existen dependencias entre ellos), es posible determinar los *codewords* escogiendo de manera aleatoria un subconjunto de vectores de características [7].

Una vez que se tiene definido los *codewords* se puede construir el *codebook* mediante el algoritmo de $k - NearestNeighbors(kNN)$ con $k = 1$:

[7]Se aprovecha que al escoger números aleatorios, éstos están uniformemente distribuidos en el rango que los define.

1: **function** CONSTRUIR CODEBOOK($n_{words}, features$)
2: $codewords = Aleatorio(features, n_{words})$
3: $codebook = kNN(codewords, k = 1)$
4: $return\ codebook$
5: **end function**

Fuente: *Elaboración propia*

Donde n_{words} es el número de *codewords* y $features$ es conjunto de vectores de características. Como $k = 1$, el *codebook* queda reducido a un diagrama de Voronoi (figura 25).

Construcción de Histogramas

La construcción de histograma se divide en dos partes:

1. **Coding:** En esta parte se mapea cada vector de características al *codeword* más cercano. Si se tiene el vector de características $v_i = \{a_1, \ldots, a_d\}, i \in 1, \ldots, n$, y el *codeword* j es el más cercano a éste, el resultado del *coding* se un vector $c_i = 0, \ldots, 1, \ldots, 0$ de tamaño n_{words}. El único 1 del vector c_i está en la posición j.

2. **Pooling:** El *Pooling* combina los resultados del *coding* para construir los histogramas. Sea C es conjunto de todos los resultados de *coding* para una imagen, se pueden definir dos tipos de *Pooling*: suma y promedio.

$$Pooling_{suma} = p1, \ldots, p_{n_{words}}/p_j = \sum c_{ij}, i \in 1, \ldots, n \tag{69}$$

$$Pooling_{promedio} = p1, \ldots, p_{n_{words}}/p_j = \frac{\sum c_{ij}}{|C|}, i \in 1, \ldots, n \tag{70}$$

Finalmente, el vector definido por *Pooling* es el histograma de la imagen. En el presente trabajo se hace uso de ambos tipos de *Pooling* y se comparan los resultados.

3.2.3.3 Clasificación

Para finalizar, en la etapa de clasificación se usan los histogramas para entrenar un *Support Vector Machine (SVM)*. Se usa este clasificador ya que [Hentschel and Sack, 2014] demostró que los SVM tiene el mejor desempeño cuando se usa BoW.

Se usa un clasificador y no un regresor porque se desea obtener la etiqueta de la clase a la cual pertenece una imagen, ésto facilita el cálculo de los resultados.

Uno de los parámetros más importantes de *SVM* es el *kernel*. Éste se define como una función de similaridad que permite calcular transformaciones de vectores de características sin que la dimensionalidad crezca exponencialmente, ésto es útil

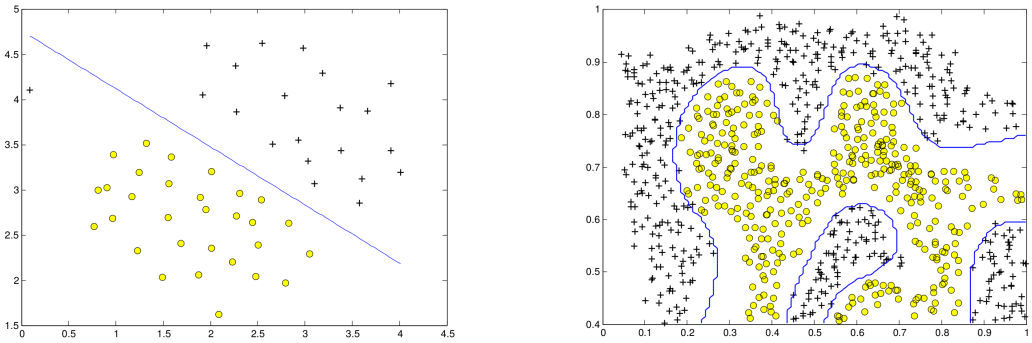
ya que permite que el *SVM* pueda predecir modelos más complejos (figura 58). Los *kernels* más populares son:

$$Kernel_{lineal}(x, x') = x^T x' \tag{71}$$

$$Kernel_{rbf}(x, x') = \exp(-\gamma \cdot |x - x'|^2) \tag{72}$$

Donde γ es un parámetro de $Kernel_{rbf}$ que controla la forma en la cual se construyen las curvas.

Figura 58: *SVM con diferentes tipos de kernel. En negro y amarillo los datos de clases diferentes, y en azul la estimación del SVM.*

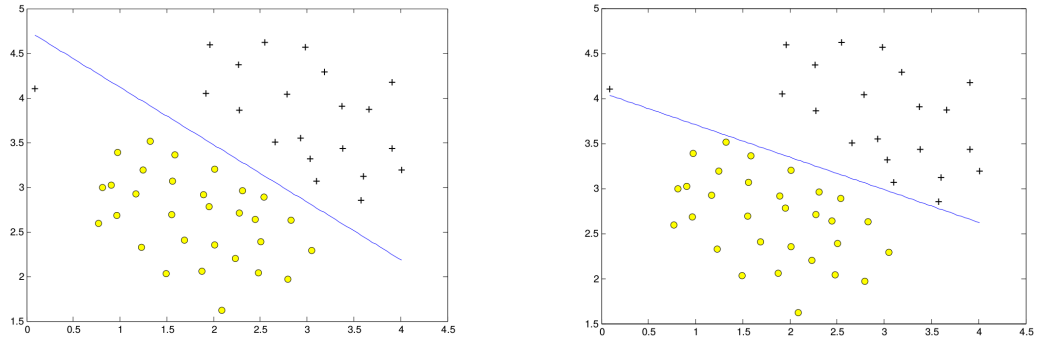


(a) kernel lineal

(b) kernel rbf

Otro parámetro importante es C que define el costo de un error en la clasificación. Un valor pequeño hace que el modelo admita falsos positivos en la curva de decisión, un valor grande hace que todos los datos esten correctamente clasificados. Ésto puede generar que el modelo estimado no sea el óptimo (figura 59).

Figura 59: *Variación de las estimaciones de SVM en base al parámetro C.*



(a) c = 1 **(b)** c = 100

Para estimar los mejores parámetros del *SVM* se realiza un proceso de búsqueda (*Grid Search*). Esta búsqueda es bastante intuitiva: Se genera un rango de valores para los parámetros C y γ (para el *kernel* rbf). Para cada combinación se realiza el entrenamiento del *SVM* sobre los datos separados en dos conjuntos (entrenamiento y evaluación), se retorna la combinación de parámetros que presentan mayor tasa de acierto.

El rango de los parámetros con los cuales se hacen las búsquedas se muestra en la tabla 6.

Tabla 6: *Parámetros usados en SVM.*

KERNEL RBF	
Parámetro	**Valor**
C	$10^i / i \in \{-5, -4, \ldots, 0, \ldots, 3, 4\}$
γ	$10^i / i \in \{-5.0, -4.5, \ldots, 0, \ldots, 3.5, 4.0\}$

KERNEL LINEAL	
Parámetro	**Valor**
C	$10^i / i \in \{-5, -4, \ldots, 0, \ldots, 3, 4\}$

3.3 Resultados

Dado que la solución del problema se divide en dos fases: Detección del danzante y Clasificación del danzante, se han desarrollado una serie de experimentos en ambas fases. En la hipótesis del trabajo se plantea una serie de variables, el análisis de la hipótesis y cada una de ellas es también realizado.

3.3.1 Detección del Danzante

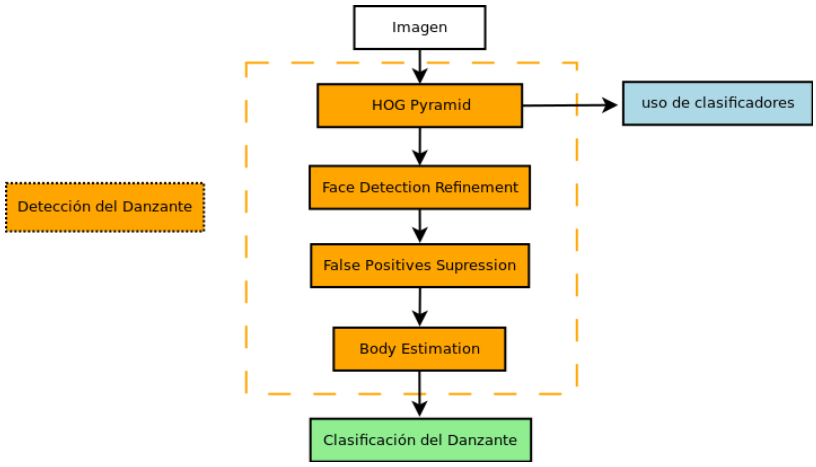
En esta sección se evalúan el desempeño de la primera fase del trabajo, además se comparan **algoritmos de clasificación** y se evalúan los efectos del **color, textura y bordes**.

3.3.1.1 Resultados experimentales

Algoritmos de Clasificación

Los métodos de clasificación son usados de la etapa de HOG Pyramid para determinar si un ventana contiene o no una máscara (figura 60), dado que la literatura no muestra trabajos previos, se evalúan el desempeño 3 algoritmos de clasificación: Adaboost, Random Forest y Support Vector Machine (SVM).

Figura 60: *Etapa en la cual son usados algoritmos de clasificación.*



Se entrenaron los 3 modelos con imágenes de tamaño 50×50 que fueron generadas a partir del 488 imágenes (19.06%) del dataset original [1], el resto de etapas de la detección del danzante son realizadas de manera normal para los 3 modelos.

[1] Se usa un porcentaje tan bajo para el entrenamiento porque el proceso de generar las imágenes iniciales es manual y con esta cantidad se generan más de 15K imágenes de 50×50 para cada clase.

Los resultados que generan cada uno de los modelos se muestran en la tabla 7. **Estos resultados son considerados los del proceso normal** ya que no se elimina ninguna etapa del proceso original, posteriormente se comparan estos resultados para medir los efectos del color, textura y bordes.

Tabla 7: *Resultados que generan en la detección los diferentes clasificadores usados en HOG Pyramid.*

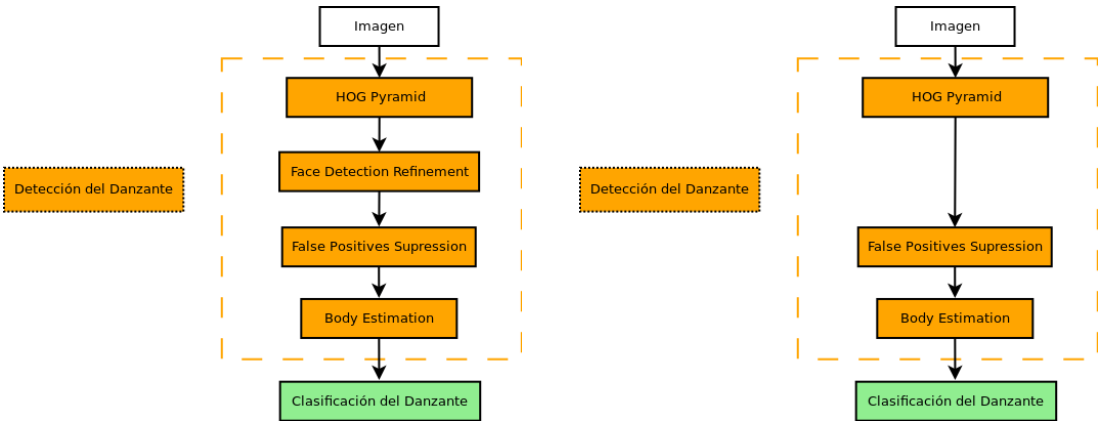
Clasificador		**Adaboost**	**Random Forest**	**SVM**
Danza	**# de imágenes**	**# de errores**	**# de errores**	**# de errores**
Qhapaq Chuncho	640	52	77	63
Qhapaq Qolla	640	36	53	56
Contradanza	640	77	106	107
Negrillo	640	27	44	105
TOTAL	**2560**	**192**	**280**	**331**
% de error	–	**7.50**	**10.94**	**12.93**

Como se puede ver el modelo de Adaboost genera mejores resultados en la detección del danzante. **Adaboost alcanza un tasa de acierto de 92.5%** mientras que Random Forest y SVM disminuyen la tasa de acierto en 3.44% y 5.43% respectivamente. Además, se observa que la danza Contradanza genera la mayor cantidad de errores, ésto se debe a que mayor cantidad de las imágenes de esta danza presentan una iluminación con muchas sombras. Ésto demuestra que a medida que las condiciones de luz se hacen más complejas los resultados de la detecciones se hacen menores. Dado que Adaboost genera menos errores, las detecciones generadas por este modelo son usadas para la fase de clasificación.

Color y Bordes

El color y los bordes son elementos principales en la etapa de Face Detection Refinement(FDR). Debido a ésto, se realizan experimentos eliminando esta etapa de la fase de detección para comparar los resultados con los del proceso normal y así evaluar los efectos que generan (figura 61), todos estos experimentos son realizados usando la salida generada por los 3 algoritmos de clasificación mostrados anteriormente.

Figura 61: *Comparación de procesos realizados para evaluar color y bordes.*



(a) Proceso Normal **(b)** Proceso sin FDR

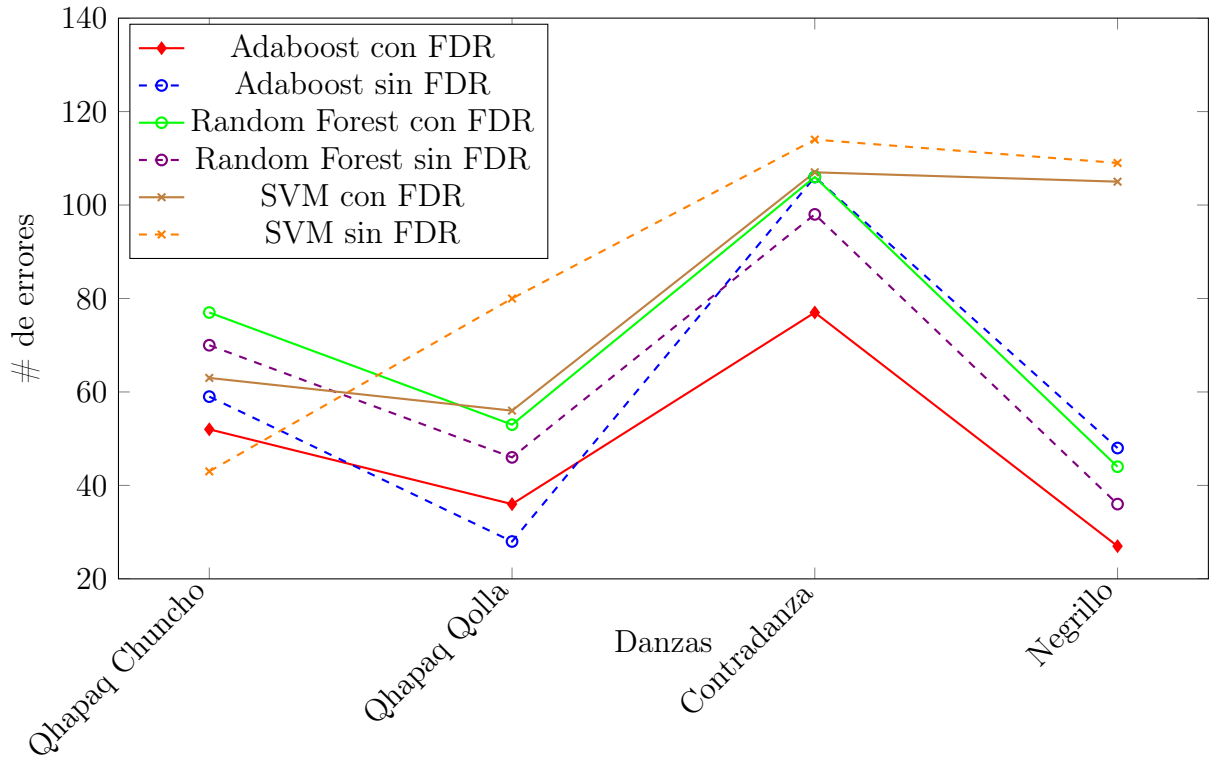
Los resultados obtenidos se muestran en la tabla 8:

Tabla 8: *Resultados que generan los bordes y el color en la detección.*

Clasificador		**Adaboost**	**Random Forest**	**SVM**
Danza	**# de imágenes**	**# de errores**	**# de errores**	**# de errores**
Qhapaq Chuncho	640	59	70	43
Qhapaq Qolla	640	28	46	80
Contradanza	640	106	98	114
Negrillo	640	48	36	109
TOTAL	**2560**	**241**	**250**	**346**
% de error	–	**9.41**	**9.77**	**13.51**

Si se compara estos resultados con los obtenidos sin eliminar Face Detection Refimenent (figura 62) se puede observar que **el color y los bordes mejoran los resultados de la detección en 1.91% cuando se usa las salidas de Adaboost y 0.58% con las de SVM, y los empeoran en 1.17% cuando se usa Random Forest**. Por otro lado, la diferencia entre la tasa de acierto de las salidas de Adaboost y Random Forest se hace mínima (0.36%), ésto indica que el color y los bordes tienen mayores efectos en la detección cuando se usan las salidas de Adaboost.

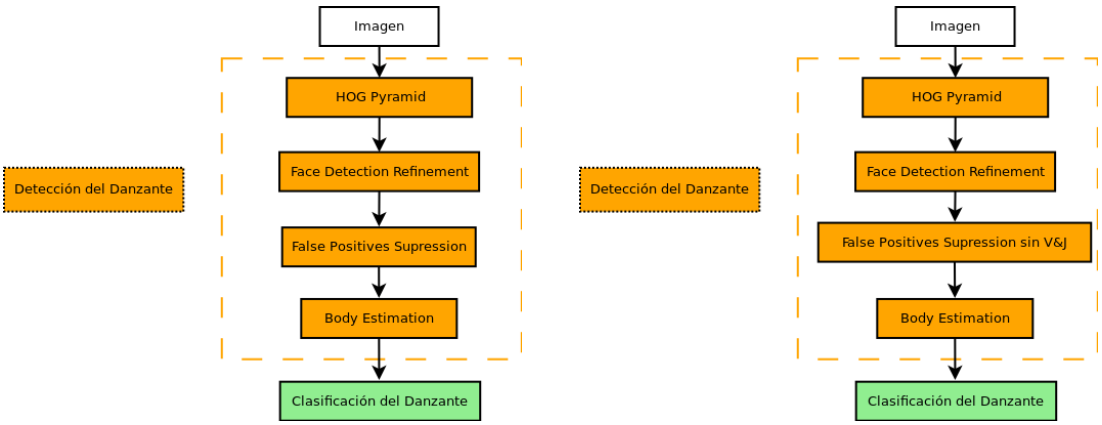
Figura 62: *Comparación de los efectos generados por el color y los bordes.*



Textura

La textura es usada como descriptor para Viola & Jones Cascade (V&J), para evaluar sus efectos en la detección, se realizan experimentos eliminando su uso dentro de la etapa de False Positives Suppression (figura 63), todos estos experimentos son realizados usando la salida generada por los 3 algoritmos de clasificación mostrados anteriormente y la etapa de Face Detection Refimenent.

Figura 63: *Comparación de procesos realizados para evaluar textura.*



(a) Proceso Normal **(b)** Proceso sin V&J

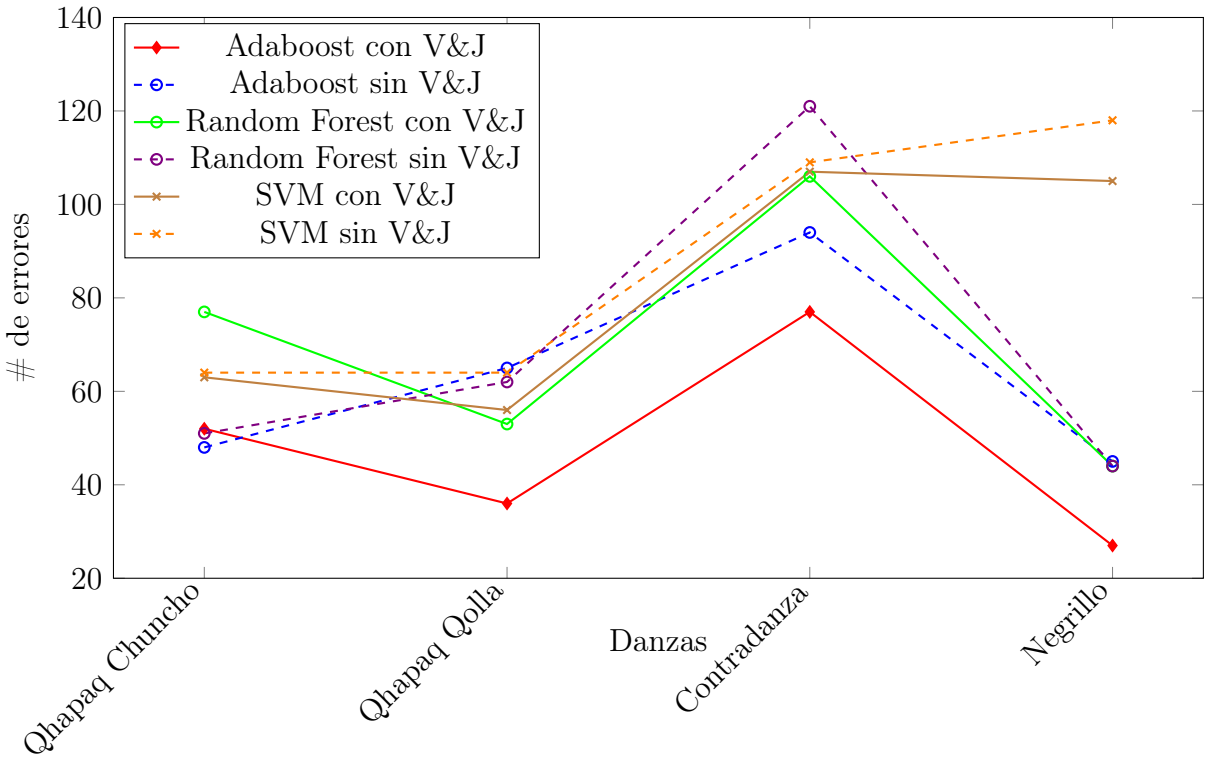
Los resultados se muestran en la tabla 9:

Tabla 9: *Resultados que genera la textura en la detección.*

Clasificador		**Adaboost**	**Random Forest**	**SVM**
Danza	**# de imágenes**	**# de errores**	**# de errores**	**# de errores**
Qhapaq Chuncho	640	48	51	64
Qhapaq Qolla	640	65	62	64
Contradanza	640	94	121	109
Negrillo	640	45	44	118
TOTAL	**2560**	**252**	**278**	**355**
% de error	–	**9.84**	**10.86**	**13.87**

Si se compara estos resultados con los obtenidos sin eliminar Viola & Jones Cascade (figura 64) se puede observar que **la textura mejora los resultados de la detección en 2.34% cuando se usa las salidas de Adaboost y 0.94% para las de SVM, y los empeoran en 0.08% para Random Forest**. Por otro lado, la diferencia entre la tasa de acierto de la detección usando las salidas de Adaboost y Random Forest disminuyen en 2.42% y, entre Adaboost y SVM disminuye en 1.40%. De igual manera se observa que las salidas de Random Forest y SVM son más invariantes a la textura.

Figura 64: *Comparación de los efectos generados por la textura.*



Los efectos generados por la textura son más relevantes que los generados por el color y los bordes.

3.3.2 Clasificación del Danzante

En esta sección se evalúan la segunda fase del trabajo, ésta tiene su eje sobre los **puntos de interés**. Para evaluar los resultados en esta fase, se hace uso de *Cross Validation*, este método divide el conjunto de datos en 2 subconjuntos, escoge uno como elemento de prueba y el otro para el entrenamiento.

Para evaluar los efectos de los puntos de interés, se comparan los vectores de características que generan *SIFT* y *SURF*, así como $Pooling_{suma}$ y $Pooling_{promedio}$ para entrenar el *SVM*. De igual manera, se definió $n_{words} \in \{50, 70, 120, 220, 320, 420, 550, 800, 1000, 1200\}$. Para aplicar *Cross Validation* se usó el 80% de las imágenes para entrenamiento y el 20% para evaluación. Los porcentajes usados para el entrenamiento y evaluación varían de trabajo en trabajo, algunos como [Krizhevsky et al., 2012] usan un 75% para entrenamiento y 25% para la evaluación, mientras otros como [Akata et al., 2014] usan más del 90% para entrenamiento y el restante para validación, en el presente trabajo se utilizan estos porcentajes porque permiten que la cantidad total de imágenes de evaluación por clase sea superior a 100, ésto permite que cada predicción influya menos de 1 punto porcentual en el resultado. Al mismo tiempo la cantidad de imágenes de entrenamiento es lo suficiente grande para construir modelos de clasificación sólidos.

3.3.2.1 Resultados experimentales

El primer conjunto de experimentos realizados, considera las fases de detección y clasificación. Por otro lado, el segundo conjunto sólo considera la fase de clasificación (figura 65).

Figura 65: *Proceso realizado para los experimentos de clasificación.*

(a) experimentos 1 **(b)** experimentos 2

Los resumen de los resultados del primer conjunto de experimentos se muestra en

la tabla 10:

Tabla 10: *Resumen del primer conjunto de experimentos.*

	SIFT					
n_{words}	**Acierto**		**Precisión**		**Exactitud**	
	$Pooling_{suma}$	$Pooling_{promedio}$	$Pooling_{suma}$	$Pooling_{promedio}$	$Pooling_{suma}$	$Pooling_{promedio}$
50		0.7988		0.80		0.80
70		0.8477		0.85		0.85
120		0.8437		0.85		0.84
220	0.8691	0.8613	0.87	0.86	0.87	0.86
320	0.8752	0.8593	0.87	0.86	0.87	0.86
420	0.8809	0.8828	0.88	0.89	0.88	0.89
550	0.8886	0.8711	0.89	0.87	0.89	0.87
800	0.8867	0.8984	0.89	0.90	0.89	0.90
1000	0.8867	0.8828	0.89	0.88	0.89	0.88
1200	0.8906	0.8809	0.89	0.88	0.89	0.88
	SURF					
n_{words}	**Acierto**		**Precisión**		**Exactitud**	
	$Pooling_{suma}$	$Pooling_{promedio}$	$Pooling_{suma}$	$Pooling_{promedio}$	$Pooling_{suma}$	$Pooling_{promedio}$
50		0.8203		0.82		0.82
70		0.8262		0.83		0.83
120		0.8652		0.87		0.87
220		0.8691		0.87		0.87
320		0.8750		0.88		0.88
420		0.8767		0.88		0.88
550		0.8828		0.88		0.88
800	0.8809	0.8828	0.88	0.88	0.88	0.88
1000	0.8730	0.8808	0.88	0.88	0.87	0.88
1200	0.8848	0.8906	0.89	0.89	0.88	0.89

El mejor resultado se obtiene se usando *SIFT*, $Pooling_{promedio}$ y $n_{words} = 800$: **tasas de acierto de 89.84%, 0.90 de Precisión y Exactitud**.

Se puede observar en las figuras 66, 67 y 68, que los resultados mejoran a medida que n_{words} crece.

De igual manera se observa que en los casos de *SURF* con $Pooling_{suma}$ y $n_{words} < 800$, *SIFT* con $Pooling_{suma}$ y $n_{words} < 220$; el *SVM* no llega a converger. Debido a ésto no se tienen resultados para estas combinaciones.

Figura 66: *Resultados para la Tasa de acierto.*

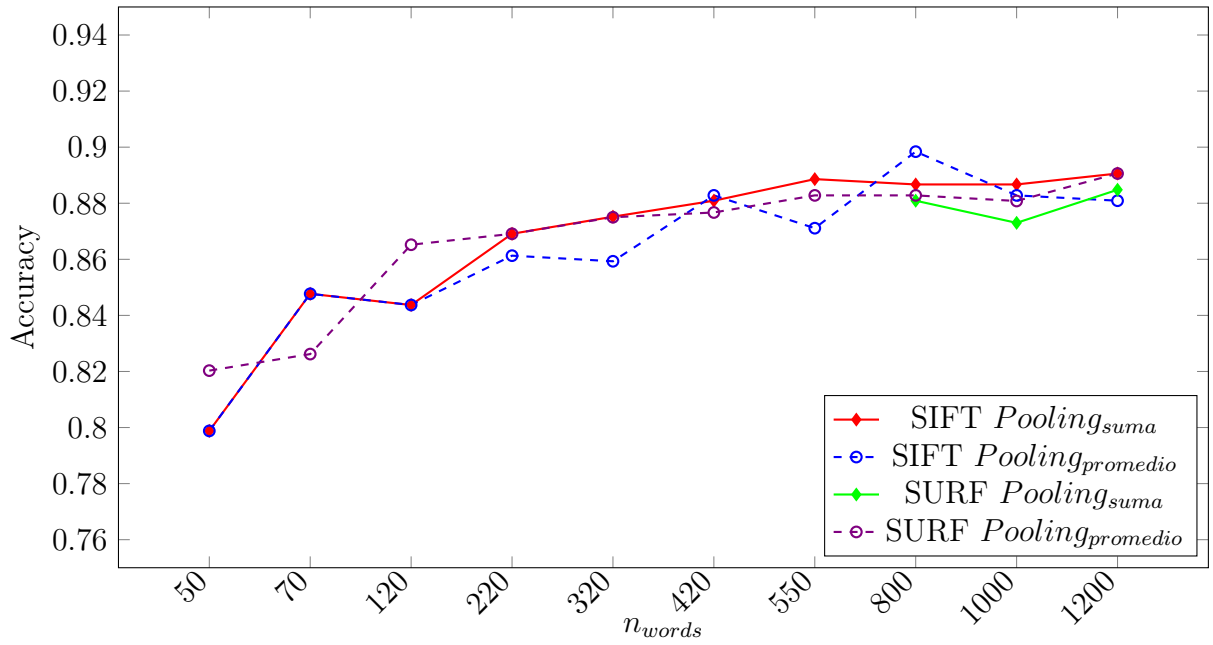


Figura 67: *Resultados para la Precisión.*

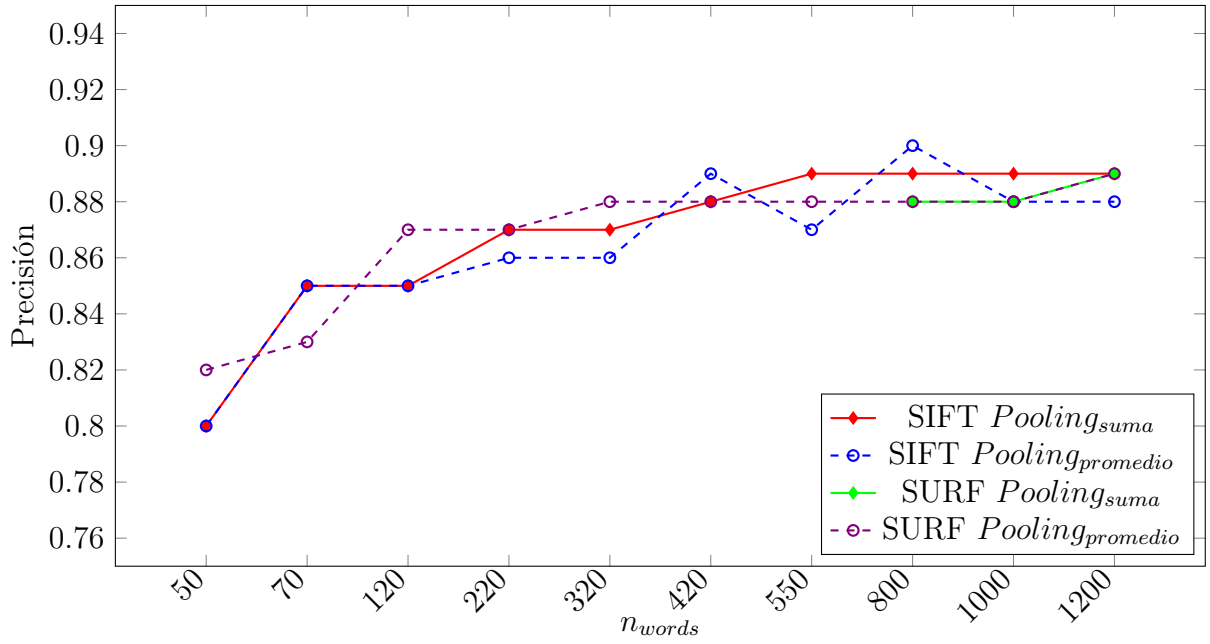
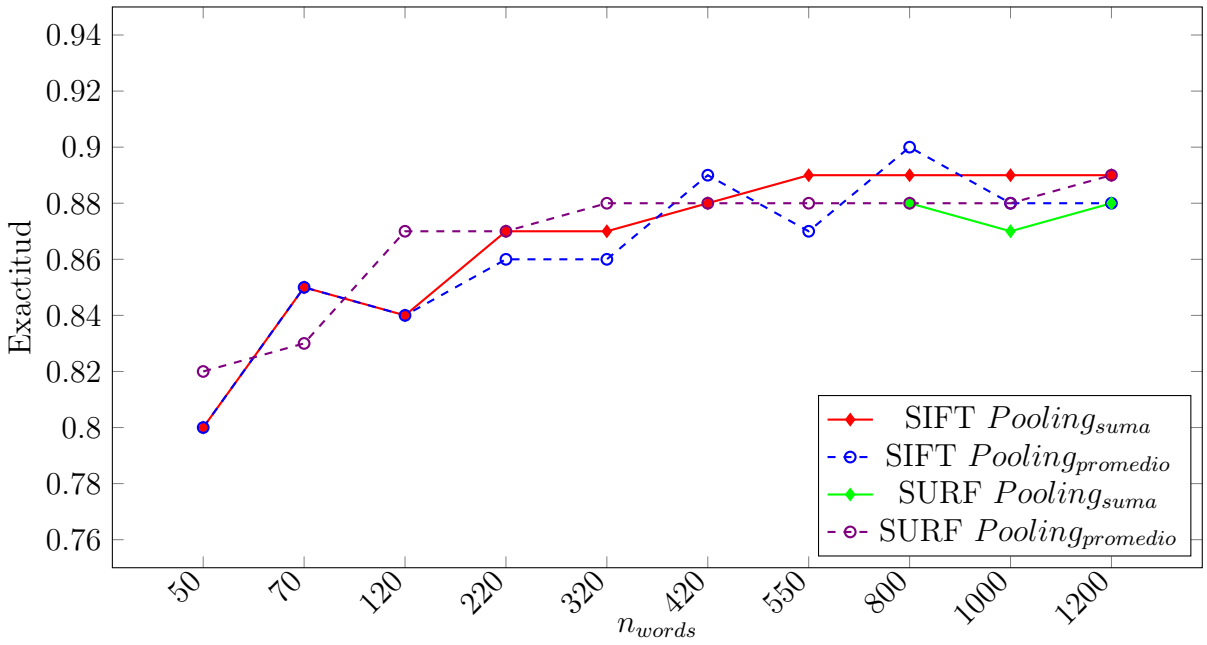


Figura 68: *Resultados para la Exactitud.*



El resumen de los resultados del segundo conjunto de experimentos es mostrado en la tabla 11

Tabla 11: *Resumen del segundo conjunto de experimentos.*

	SIFT					
n_{words}	**Acierto**		**Precisión**		**Exactitud**	
	$Pooling_{suma}$	$Pooling_{promedio}$	$Pooling_{suma}$	$Pooling_{promedio}$	$Pooling_{suma}$	$Pooling_{promedio}$
50		0.8262		0.83		0.83
70		0.8359		0.84		0.84
120		0.8633		0.86		0.86
220	0.8770	0.8887	0.88	0.86	0.88	0.86
320	0.8887	0.8809	0.89	0.89	0.89	0.88
420	0.8926	0.8770	0.89	0.88	0.89	0.88
550	0.8828	0.8828	0.89	0.88	0.88	0.88
800	0.8730	0.8770	0.87	0.88	0.87	0.88
1000	0.8730	0.8613	0.87	0.86	0.87	0.86
1200	0.8789	0.8730	0.88	0.88	0.88	0.87
	SURF					
n_{words}	**Acierto**		**Precisión**		**Exactitud**	
	$Pooling_{suma}$	$Pooling_{promedio}$	$Pooling_{suma}$	$Pooling_{promedio}$	$Pooling_{suma}$	$Pooling_{promedio}$
50		0.7773		0.78		0.78
70		0.8027		0.80		0.80
120		0.8301		0.83		0.83
220	0.8594	0.8652	0.86	0.87	0.86	0.87
320	0.8750	0.8750	0.88	0.88	0.88	0.88
420	0.8750	0.8828	0.88	0.89	0.88	0.89
550	0.8887	0.8945	0.89	0.90	0.89	0.89
800	0.8633	0.8711	0.87	0.87	0.86	0.87
1000	0.8770	0.8789	0.88	0.88	0.88	0.88
1200	0.8848	0.9023	0.89	0.90	0.88	0.90

El mejor resultado se obtiene se usando *SURF*, $Pooling_{promedio}$ y $n_{words} = 1200$: **tasas de acierto de 90.23%, 0.90 de Precisión y Exactitud** .

Se puede observar en las figuras 69, 70 y 71, que al igual de los experimentos con detección del danzante, los resultados mejoran a medida que n_{words} crece.

De igual manera se observa que en los casos de *SURF* con $Pooling_{suma}$ y $n_{words} < 220$, *SIFT* con $Pooling_{suma}$ y $n_{words} < 220$; el *SVM* no llega a converger. Debido a ésto no se tienen resultados para estas combinaciones.

Figura 69: *Resultados para la Tasa de acierto eliminando detección del danzante.*

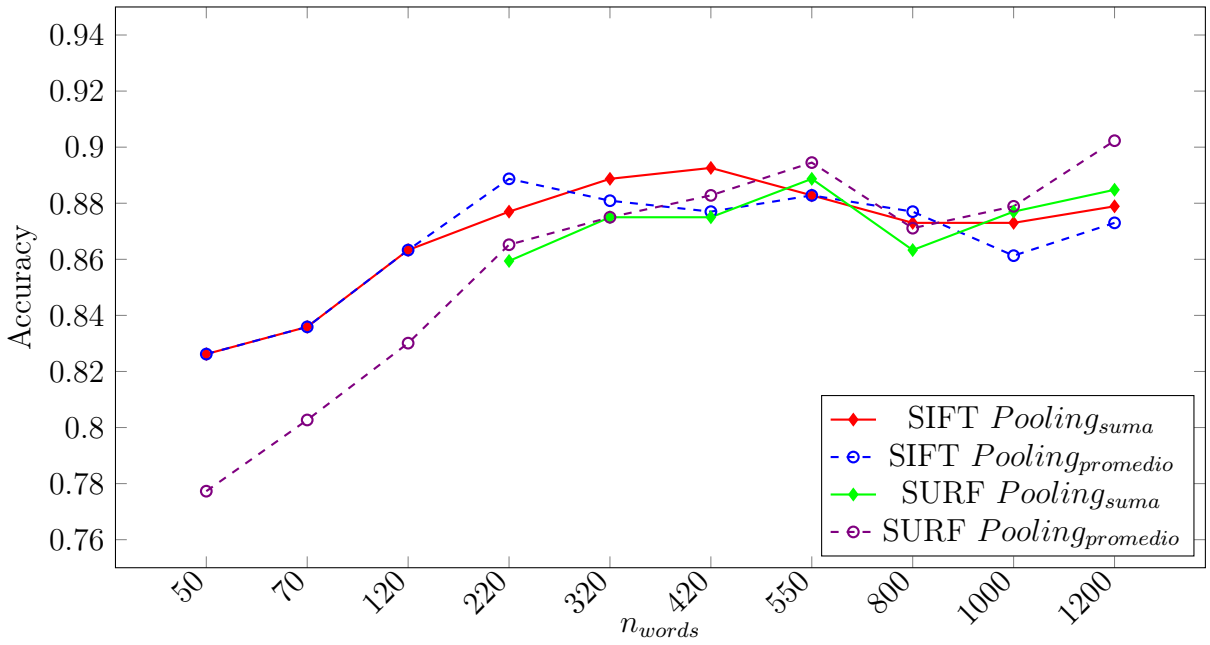


Figura 70: *Resultados para la Precisión eliminando detección del danzante.*

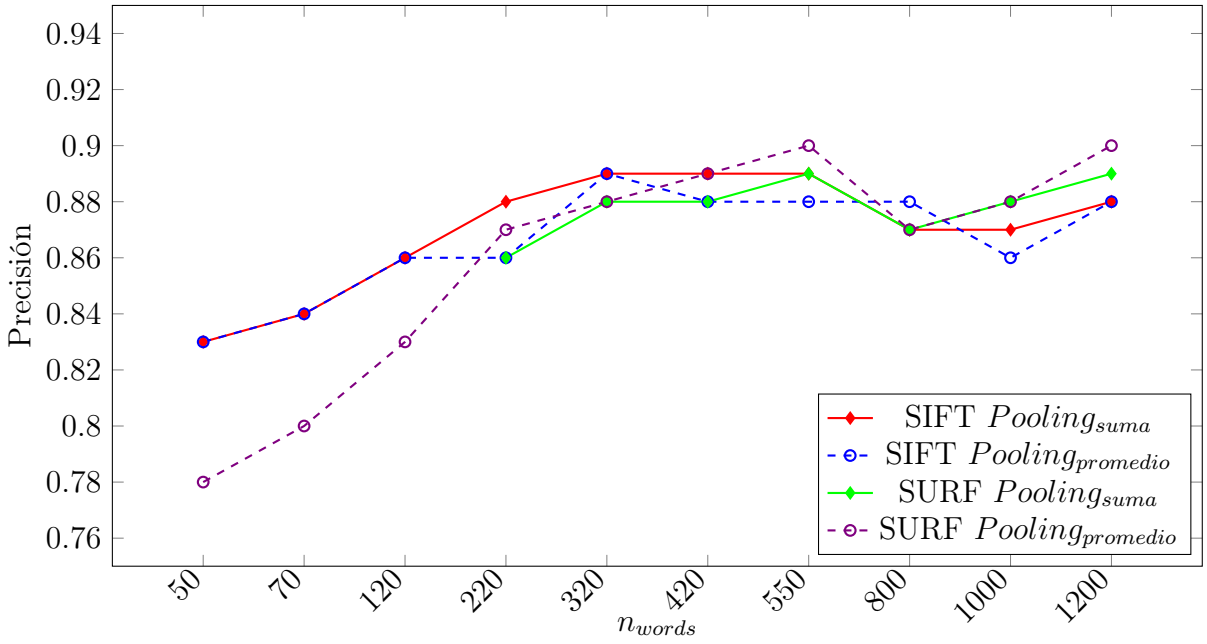
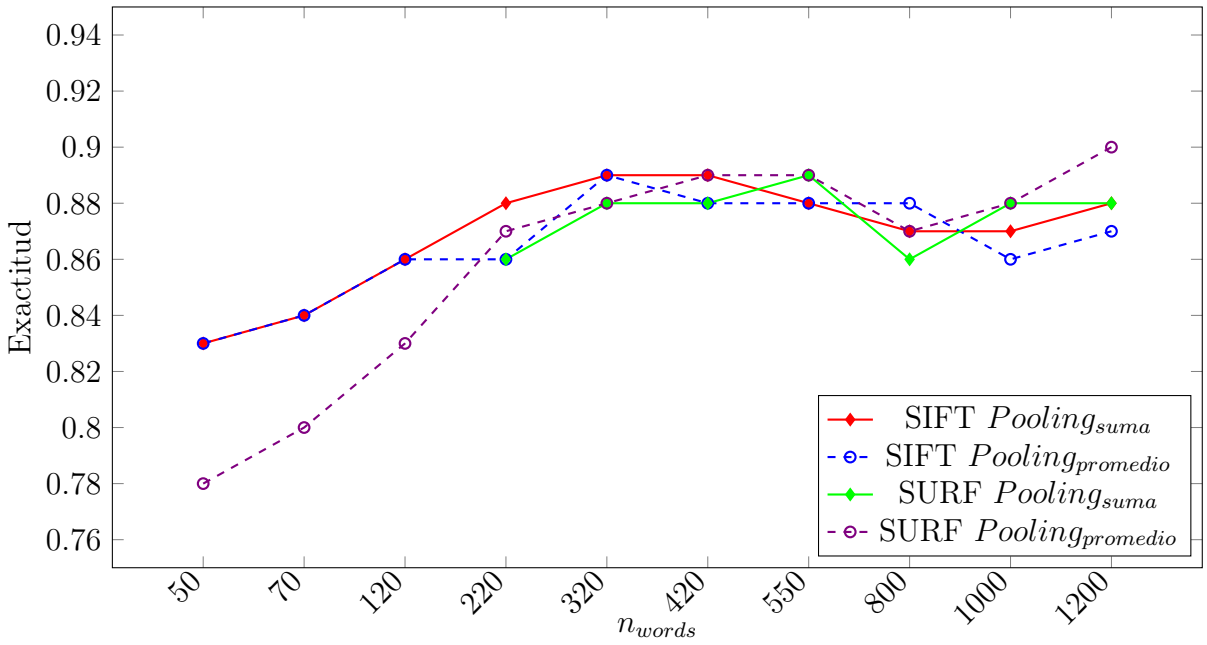


Figura 71: *Resultados para la Exactitud eliminando detección del danzante.*

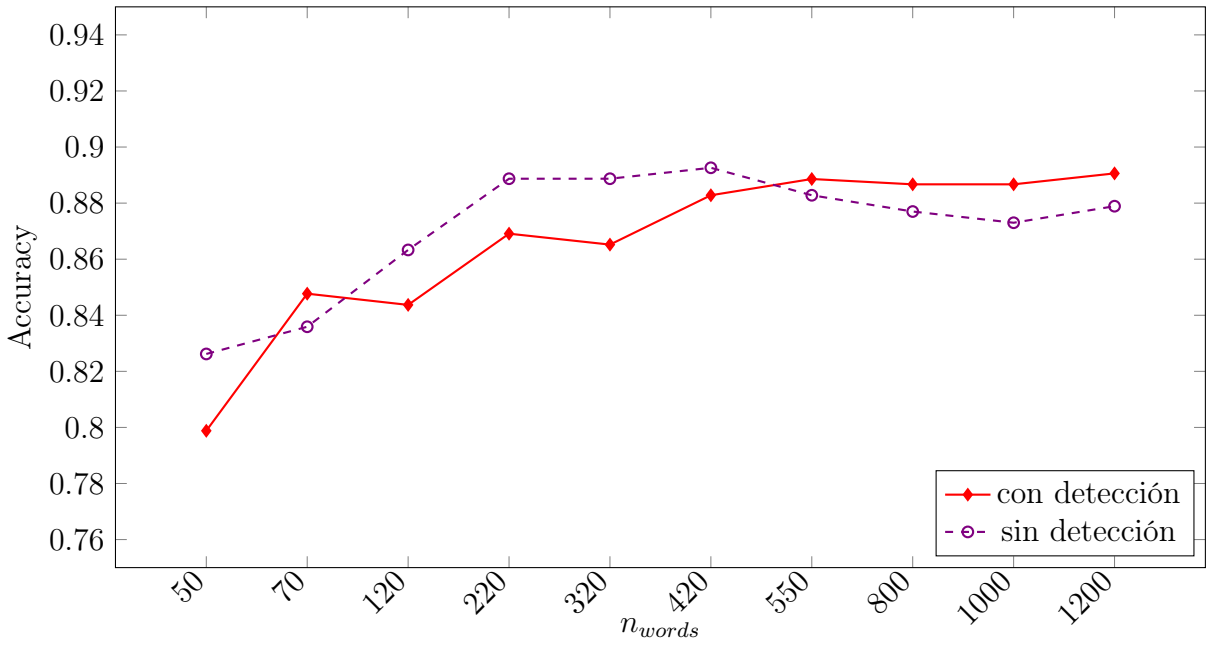


En la figura 72 y tabla 12 se muestra la comparación de los mejores resultados de tasa de acierto, con *SIFT*, entre la clasificación con y sin detección del danzante. Se puede observar que a medida que n_{words} crece, la clasificación con detección del danzante se hace mejor.

Tabla 12: *Comparación de mejores resultados para SIFT entre la clasificación con y sin detección del danzante.*

SIFT		
n_{words}	**Con detección**	**Sin detección**
50	0.7988	0.8262
70	0.8477	0.8359
120	0.8437	0.8633
220	0.8691	0.8887
320	0.8652	0.8887
420	0.8828	0.8926
550	0.8886	0.8828
800	0.8867	0.8711
1000	0.8867	0.8770
1200	0.8906	0.8789

Figura 72: *Comparación de mejores resultados para SIFT entre la clasificación con y sin detección del danzante.*

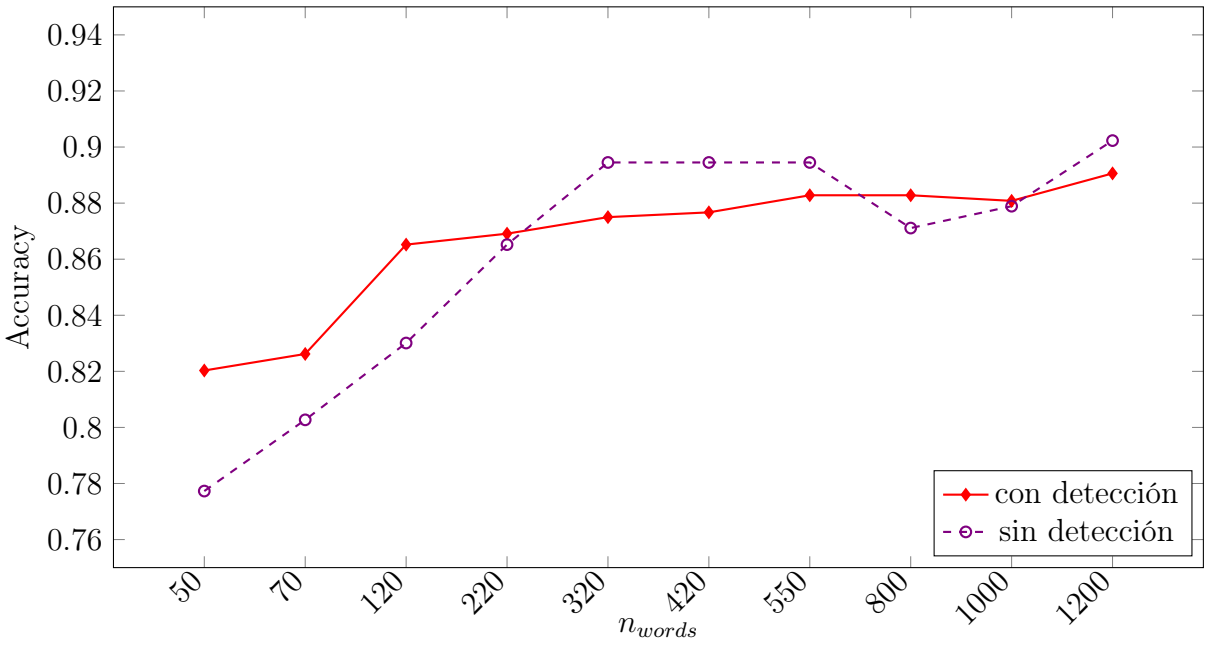


En la figura 73 y tabla 13 se muestra la comparación de los mejores resultados de tasa de acierto, con *SURF*, entre la clasificación con y sin detección del danzante. Se puede observar que la clasificación con detección del danzante es mejor para los valores más pequeños de n_{words}, en valores intermedios se torna peor y finalmente mejora (a excepción de $n_{words} = 1200$).

Tabla 13: *Comparación de mejores resultados para SURF entre la clasificación con y sin detección del danzante.*

SURF		
n_{words}	**Con detección**	**Sin detección**
50	0.8203	0.7773
70	0.8262	0.8027
120	0.8652	0.8301
220	0.8691	0.8652
320	0.8750	0.8945
420	0.8767	0.8945
550	0.8828	0.8945
800	0.8828	0.8711
1000	0.8808	0.8789
1200	0.8906	0.9023

Figura 73: *Comparación de mejores resultados para SURF entre la clasificación con y sin detección del danzante.*



De las figuras 72 y 73 se puede observar que la etapa de detección al danzante no genera mejoras significativas en los resultados de Tasa de acierto, Precisión y Exactitud. Ésto se debe a los siguientes factores:

1. **Descriptores:** Dado que se usan descriptores locales y que las imágenes resultado de la fase de detección del danzante son subregiones de las imágenes originales, los puntos de interés son los mismos en la subregión. En el caso de las imágenes originales también se tiene puntos de interés en áreas que no son relevantes (fondo de la imagen), éstos no juegan un papel importante en los resultados por la forma en la cual se entrena el *SVM* (factor 3, parámetros de SVM).

2. **Resultados de la fase de detección:** Como se observa en los resultados de la fase de detección, existe una tasa de error de 7.5%, la mayoría de estos errores sólo detectan el cuerpo (85.02%). Dado que los resultados de esta fase son la entrada de la fase de clasificación, el error se propaga generando que los efectos de usar la detección no sean relevantes.

3. **Parámetros de SVM:** Dado que para estimar los mejores parámetros de *SVM* se realizan búsquedas en un espacio de parámetros predefinido y que cada una de estas combinación de parámetros se evalúa separando datos para entrenamiento y evaluación, las mejores estimaciones se obtiene con un parámetro C pequeño (este parámetro indica el costo por error de un *SVM*). Al ser C pequeño, los

puntos de interés que están fuera del danzante no juegan un papel relevante en la construcción del *SVM* y por tanto en los resultados finales. En las tablas 14 y 15 se pueden observar los parámetros que generan los mejores resultados para la clasificación usando SURF.

Tabla 14: *Resumen de resultados con SURF junto a parámetros que los generan.*

Con detección del danzante					
n_{words}	**Acierto**	**Pooling**	**kernel**	C	γ
50	0.8203	promedio	rbf	10	100.0
70	0.8262	promedio	rbf	10	316.22
120	0.8652	promedio	rbf	10	100.0
220	0.8691	promedio	rbf	10	100.0
320	0.8750	promedio	rbf	100	316.22
420	0.8767	promedio	rbf	10	100.0
550	0.8828	promedio	rbf	100	316.22
800	0.8828	promedio	rbf	100	316.22
1000	0.8808	promedio	rbf	10000	0.0316
1200	0.8906	promedio	rbf	10	316.2

Tabla 15: *Resumen de resultados con SURF junto a parámetros que los generan.*

Sin detección del danzante					
n_{words}	**Acierto**	**Pooling**	**kernel**	C	γ
50	0.7773	promedio	rbf	10	316.22
70	0.8027	promedio	rbf	1000	31.62
120	0.8301	promedio	rbf	10	100
220	0.8652	promedio	rbf	100	316.22
320	0.8945	promedio	rbf	10	316.22
420	0.8945	promedio	rbf	100	316.22
550	0.8945	promedio	rbf	10	316.22
800	0.8711	promedio	rbf	100	316.22
1000	0.8789	promedio	rbf	10	316.22
1200	0.9023	promedio	rbf	10	1000.0

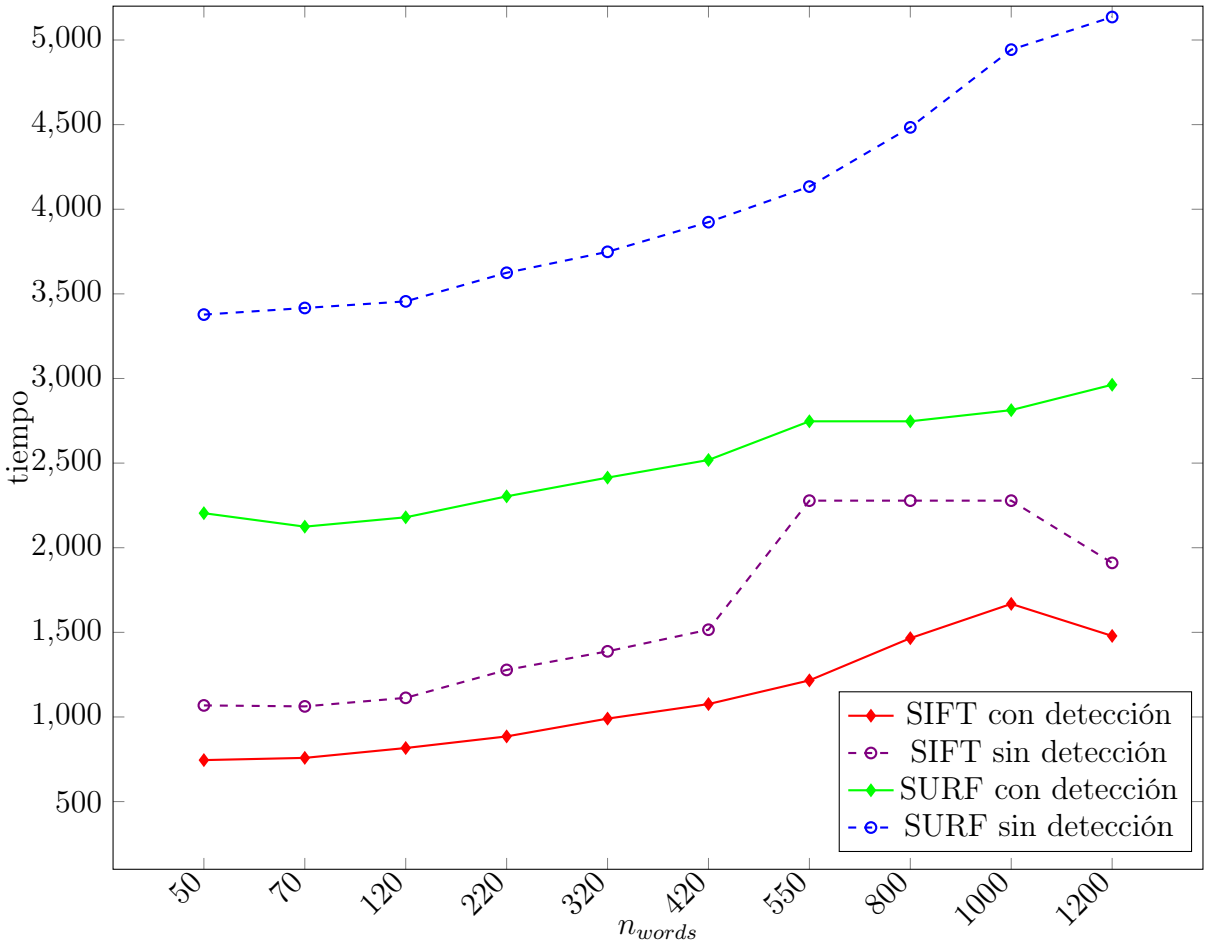
Si bien los efectos de la detección en la tasa de acierto, precisión y exactitud son mínimos. Los efectos en el uso de memoria (tabla 16) y tiempo de procesamiento (figura 74) son interesantes. Ésto se debe a que al detectar los danzantes, el tamaño total de los datos de entrada para la clasificación es menor y por ende la cantidad de vectores de características y tiempo de entrenamiento.

Tabla 16: *Uso de memoria promedio en Gb.*

SIFT	
Sin detección	**Con detección**
7.9	3.9
Mejora en 50.63 %	

SURF	
Sin detección	**Con detección**
14.6	11.9
Mejora en 18.49%	

Figura 74: *Comparación de mejores tiempos (en segundos) de entrenamiento para SIFT y SURF.*



Si bien, a medida que n_{words} crece mejoran los resultados, el tiempo de ejecución también aumenta.

Mediante el uso de la etapa de detección, el tiempo de entrenamiento para la clasificación mejora en **8.2 horas** mientras que el uso de memoria promedio disminuye en **6.7 Gb**.

3.3.3 Detalles Técnicos

Para llevar a cabo el proyecto se hizo uso del siguiente hardware y software.

3.3.3.1 Hardware

- **Cámara:** Nikon D7200.
- **Notebook:** Toshiba Satellite S845. Procesador Intel Core-i5 3rd generation/2.5 GHz, 4Gb RAM DDR3.
- **Notebook:** Lenovo Y700. Procesador Intel Core-i7 6rd generation/3.5 GHz, 8Gb RAM DDR4.

3.3.3.2 Software

- **Sistema Operativo:** Ubuntu 16.04.1 LTS x86_64 .
- **Lenguaje de Programación:** Python 2.7 .
- **Librerías:** Opencv 3.0.0, NumPy 1.11.2, SciPy 0.17.0, SciKitLearn 0.18 y SciKitImage 0.12.3.
- **Controlador de versiones:** Git 2.10 .

Conclusiones

1. Las conclusiones en base al objetivo general son:

   (a) Se logra una tasa de acierto 92.50% para la detección de danzantes y 90.23% para la clasificación, en base a ésto y el proceso de contraste, la hipótesis planteada en el proyecto es aceptada.

   (b) Se propuso un método para detectar danzantes y clasificar imágenes de danzas típicas del Cusco, mediante el uso de HOG Pyramid, LBP, SIFT, SURF, Eigenfaces, Fisherfaces, Viola & Jones Cascade, Adaboost, Random Forest, SVM y Bag of Words, obteniendo una tasa de acierto final de 90.23%, una precisión de 0.9 y exactitud de 0.9. Ésto demuestra la capacidad del Machine Learning en el problema de clasificación.

   (c) El enfoque propuesto realiza una etapa de detección del danzante, ésta ha demostrado no tener efectos relevantes en la clasificación dentro del dataset creado, pero si en los recursos computacionales necesarios para el entrenamiento.

2. La conclusión en base al primer objetivo específico es:

   (a) Las diferencias entre el desempeño de SIFT y SURF para la extracción de características en imágenes de danzas son mínimas. Mientras que el kernel rbf ha demostrado tener mejores resultados que el kernel lineal.

3. La conclusión en base al segundo objetivo específico es:

   (a) Se construyó un dataset de 2560 imágenes que corresponden a 4 danzas del Cusco [2].

4. Las conclusiones en base al tercer objetivo específico son:

   (a) Adaboost genera mejores resultados que Random Forest y SVM en la detección del danzante. Por otro lado, SVM genera resultados altos en la clasificación del danzante.

   (b) Con determinadas combinaciones de n_{words} y *polling*, los puntos de interés representados por vectores de características pueden convertirse en histogramas muy parecidos para clases diferentes. Ésto hace que el SVM no logre converger.

[2] Las imágenes junto a los modelos entrenados se encuentran en `https://goo.gl/NDxXk4`

5. Las conclusiones en base al cuarto objetivo específico son:

   (a) El uso de *sliding windows* genera un cuello de botella debido a su complejidad computacional, $O(nmx)$ donde n, m son las dimensiones la imagen y x es el costo de la operación sobre la ventana.

   (b) Los modelos de detección de rostros entrenados en máscaras permiten tener un *recall* alto.

   (c) La mayor variación generada por el color y los bordes es de un 1.91%, ésto demuestra que sus efectos no son muy relevantes.

   (d) La mayor variación generada por la textura es de un 2.43%, ésto demuestra que este factor es más importante que el color y los bordes en la detección del danzante.

   (e) Los puntos de interés demuestran ser un factor vital en la clasificación del danzante ya que permiten obtener tasas de acierto altas.

   (f) La implementación fue realizada usando Python como lenguaje principal [3]. Sin embargo, se también se usaron librerías y herramientas externas[4] implementadas en C++, Python y Perl:

      - En la etapa de HOG Pyramid se hizo uso de las implementaciones de Adaboost, Random Forest y SVM de la librería SciKitLearn, de igual forma, se implementó Integral Image para poder calcular la función ξ en $O(1)$.
      - Face Detection Refinement fue implementada usando la librería SciKitImage para los filtros y Hough Transform.
      - Para la etapa de False Positives Suppression se usó las implementaciones de Fisherfaces, Eigenfaces y Viola & Jones Cascade de OpenCV, además, en el caso de Viola & Jones Cascade se usó la herramienta externa implementada en Perl [5]
      - La implementación de Body Estimation fue realizada usando NumPy para poder vectorizar las operaciones.
      - En la implementación de Bag of Words se usó SciKitLearn para paralelizar el proceso de cálculo de histogramas y Grid Search.

[3] El repositorio del proyecto se encuentra disponible en `https://github.com/RodolfoQuispeC/Dance-Recognition`

[4] Las librerías se listan en la sección 3.3.3.

[5] Ésta se encuentra disponible en `https://github.com/mrnugget/opencv-haar-classifier-training` y fue usada para la construcción de los *samples* usados en el entrenamiento.

Recomendaciones

1. Se deja para trabajos futuros el desarrollo de enfoques para la detección del danzante que no usen las máscaras como eje principal. Éstos deben solucionar el problema de dada una imagen ¿Contiene ésta un danzante?.

2. El deep learning ha demostrado ser altamente discriminativo en problemas de clasificación por lo que su uso en este problema debería tener resultados superiores al obtenido.

3. Trabajos futuros deben plantear soluciones para la clasificación de danzas multietiquetada.

4. El uso de móviles representa una oportunidad para el desarrollo de aplicaciones que permitan mostrar información relevante sobre danzas así como la creación de *datasets* más grandes y complejos. Esta aplicación debe considerar el almacenamiento y procesamiento de las imágenes en un servicio remoto ya que la implementación actual requiere un alto poder de cómputo.

5. La evaluación de imágenes en lugares cerrados (teatros, restaurantes, etc) y con luz artificial debería ser evaluado en futuros trabajos.

6. Implementaciones futuras deben considerar la paralelización del proceso de *sliding windows* para mejorar el tiempo de respuesta del algoritmo.

Bibliografía

[Abo-Zahhad et al., 2014] Abo-Zahhad, M., Gharieb, R. R., Ahmed, S. M., Donkol, A. A. E.-B., et al. (2014). Edge detection with a preprocessing approach. *Journal of Signal and Information Processing*, 5(04):123.

[Akata et al., 2014] Akata, Z., Perronnin, F., Harchaoui, Z., and Schmid, C. (2014). Good practice in large-scale learning for image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):507–520.

[Avila et al., 2013] Avila, S., Thome, N., Cord, M., Valle, E., and AraúJo, A. D. A. (2013). Pooling in image representation: The visual codeword point of view. *Computer Vision and Image Understanding*, 117(5):453–465.

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[Bradski, 2000] Bradski, G. (2000). Opencv. *Dr. Dobb's Journal of Software Tools*.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, 60(6):679–698.

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE.

[de los Santos Y., 2010] de los Santos Y., A. (2010). La teoría del color. Fundamentos Visuales 2, IDAT.

[Escalera et al., 2015] Escalera, S., Fabian, J., Pardo, P., Baró, X., Gonzalez, J., Escalante, H. J., Misevic, D., Steiner, U., and Guyon, I. (2015). Chalearn looking at people 2015: Apparent age and cultural event recognition datasets and results. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1–9.

[Everingham et al., 2010] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338.

[Felzenszwalb et al., 2010] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645.

[Fisher et al., 1994] Fisher, R., Perkins, S., Walker, A., and Wolfart, E. (1994). Hypermedia image processing reference. *Department of Artificial Intelligence, University of Edinburgh*.

[Fisher, 1936] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188.

[Fletcher, 2009] Fletcher, T. (2009). Support vector machines explained. University College London.

[Girshick et al., 2010] Girshick, R. B., Felzenszwalb, P. F., and McAllester, D. (2010). Discriminatively trained deformable part models, release 5. http://people.cs.uchicago.edu/ rbg/latent-release5/.

[Haralick et al., 1973] Haralick, R. M., Shanmugam, K., et al. (1973). Textural features for image classification. *IEEE Transactions on systems, man, and cybernetics*, 24(6):610–621.

[Hentschel and Sack, 2014] Hentschel, C. and Sack, H. (2014). Does one size really fit all?: Evaluating classifiers in bag-of-visual-words classification. In *Proceedings of the 14th International Conference on Knowledge Technologies and Data-driven Business*, page 7. ACM.

[Hernández Sampieri et al., 2010] Hernández Sampieri, C. R., Fernández Collado, C., and Baptista Lucio, P. (2010). *Metodología de la Investigación*. MCGRAW-HILL.

[Hinton, 2002] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800.

[Kapsouras et al., 2013] Kapsouras, I., Karanikolos, S., Nikolaidis, N., and Tefas, A. (2013). Feature comparison and feature fusion for traditional dances recognition. In *International Conference on Engineering Applications of Neural Networks*, pages 172–181. Springer.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Le et al., 2011] Le, Q. V., Zou, W. Y., Yeung, S. Y., and Ng, A. Y. (2011). Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3361–3368. IEEE.

[Leung, 2004] Leung, K. M. (2004). k-nearest neighbor algorithm for classification. Polytechnic University Department of Computer Science / Finance and Risk Engineering.

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110.

[Maio and Maltoni, 2000] Maio, D. and Maltoni, D. (2000). Real-time face location on gray-scale static images. *Pattern Recognition*, 33(9):1525–1539.

[Martinez, 2011] Martinez, A. (2011). Fisherfaces. *Scholarpedia*, 6(2):4282.

[Matas and Sochman, 2004] Matas, J. and Sochman, J. (2004). Adaboost. Centre for Machine Perception, Czech Technical University.

[Mejía, 2005] Mejía, E. M. (2005). *Metodología de la Investigación Científica*. Universidad Nacional Mayor de San Marcos.

[Nussipbekov et al., 2014] Nussipbekov, A., Amirgaliyev, E., and Hahn, M. (2014). Kazakh traditional dance gesture recognition. In *Journal of Physics: Conference Series*, volume 495, page 012036. IOP Publishing.

[O'Brien, 1997] O'Brien, D. (1997). Cassini lossy compression software tests.

[Ojala et al., 2002] Ojala, T., Pietikainen, M., and Maenpaa, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987.

[Otsu, 1975] Otsu, N. (1975). A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27.

[Panchal et al., 2013] Panchal, P., Panchal, S., and Shah, S. (2013). A comparison of sift and surf. *International Journal of Innovative Research in Computer and Communication Engineering*, 1(2):323–327.

[Papadopoulos et al., 2011] Papadopoulos, S., Troncy, R., Mezaris, V., Huet, B., and Kompatsiaris, I. (2011). Social event detection at mediaeval 2011: Challenges, dataset and evaluation. In *MediaEval*.

[Papadopoulos et al., 2012] Papadopoulos, S., Troncy, R., Mezaris, V., Schinas, E., and Kompatsiaris, I. (2012). Social event detection at mediaeval 2012: Challenges, datasets, and evaluation. In *MediaEval*.

[Parekh et al., 2014] Parekh, H. S., Thakore, D. G., and Jaliya, U. K. (2014). A survey on object detection and tracking methods. *International Journal of Innovative Research in Computer and Communication Engineering*, 2(2):2970–2979.

[Paris et al., 2009] Paris, S., Kornprobst, P., Tumblin, J., and Durand, F. (2009). *Bilateral filtering: Theory and applications*. Now Publishers Inc.

[Parker, 2011] Parker, J. (2011). *Algorithms for Image Processing and Computer Vision*. Wiley Publishing, Inc.

[Peng et al., 2008] Peng, B., Qian, G., and Peng, B. (2008). Binocular dance pose recognition and body orientation estimation via multilinear analysis. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–8. IEEE.

[Pentland et al., 1994] Pentland, A., Moghaddam, B., and Starner, T. (1994). View-based and modular eigenspaces for face recognition. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 84–91. IEEE.

[Petkos et al., 2014a] Petkos, G., Papadopoulos, S., Mezaris, V., and Kompatsiaris, Y. (2014a). Social event detection at mediaeval 2014: Challenges, datasets, and evaluation. In *MediaEval*.

[Petkos et al., 2014b] Petkos, G., Papadopoulos, S., Mezaris, V., Troncy, R., Cimiano, P., Reuter, T., and Kompatsiaris, Y. (2014b). Social event detection at mediaeval: a three-year retrospect of tasks and results. In *ICMR 2014 Workshop on Social Events in Web Multimedia (SEWM)*, pages 27–34.

[Reuter et al., 2013] Reuter, T., Papadopoulos, S., Petkos, G., Mezaris, V., Kompatsiaris, Y., Cimiano, P., De Vries, C. M., and Geva, S. (2013). Social event detection at mediaeval 2013: Challenges, datasets, and evaluation. In *MediaEval*.

[Samanta and Chanda, 2013] Samanta, S. and Chanda, B. (2013). A novel technique for space-time-interest point detection and description for dance video classification. In *International Symposium on Visual Computing*, pages 507–516. Springer.

[Samanta and Chanda, 2014] Samanta, S. and Chanda, B. (2014). Indian classical dance classification on manifold using jensen-bregman logdet divergence. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 4507–4512. IEEE.

[Samanta et al., 2012] Samanta, S., Purkait, P., and Chanda, B. (2012). Indian classical dance classification by learning dance pose bases. In *Applications of Computer Vision (WACV), 2012 IEEE Workshop on*, pages 265–270. IEEE.

[Samuel, 1959] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.

[Silvela and Portillo, 2001] Silvela, J. and Portillo, J. (2001). Breadth-first search and its application to image processing problems. *IEEE Transactions on Image Processing*, 10(8):1194–1199.

[Tomasi and Manduchi, 1998] Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846. IEEE.

[Torres, 2014] Torres, B. (2014). Nuevo enfoque para la segmentación de texto en fotografías que contengan señales informativas direccionales de las calles de cusco basado en k-means y disjoint sets. Universidad Nacional de San Antonio Abad del Cusco, Cusco, Perú.

[Van der Walt et al., 2014] Van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., and the scikit-image contributors (2014). scikit-image: image processing in Python. *PeerJ*, 2:e453.

[Viola and Jones, 2001] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE.

[Zhang and Turk, 2008] Zhang, S. and Turk, M. (2008). Eigenfaces. *Scholarpedia*, 3(9):4244.

[Zhu and Ramanan, 2012] Zhu, X. and Ramanan, D. (2012). Face detection, pose estimation, and landmark localization in the wild. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2879–2886. IEEE.